Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Petrovčič

Referenčna arhitektura za razvoj interoperabilnih rešitev na področju interneta stvari

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Marko Bajec

Ljubljana, 2016

Zahvaljujem se prof. dr Marku Bajcu za vse strokovne nasvete, ideje in pomoč med pisanjem magistrske naloge. Za lektoriranje naloge se zahvaljujem lektorici Tjaši Skazi. Iskreno se zahvaljujem tudi domačim in najbližjim za vso podporo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
IoT	Internet of Things	Internet stvari
OM2M	Open Machine To Machine	Platforma OM2M
HTTP	Hyper Text Transport Protocol	Protokol za prenos hiperteksta
CoAP	COnstrained Application Protocol	Protokol za naprave z omejenimi sistemskimi viri
CSE	Common Service Entity	Entiteta standardnih storitev
AE	Application entity	Aplikacijska entiteta
IN	Infrastructure Node	Vozlišče lokalne infrastrukture
IN-CSE	Infrastructure Node - CSE	Glavno vozlišče, ki vsebuje CSE
MN-CSE	Middle Node - CSE	Vmesno vozlišče, ki vsebuje CSE
IPU	Interworking Proxy Unit	Vtičnik platforme OM2M
OBiX	Open Building Information Xchange	Standard za opisovanje stavb z uporabo formata XML
AT	ATtention	Ukazi AT

Povzetek

Naslov: Referenčna arhitektura za razvoj interoperabilnih rešitev na področju interneta stvari

V nalogi je predstavljena referenčna arhitektura interneta stvari, sestavljena iz strojnega in programskega dela, ki omogoča komuniciranje naprav po trenutno najbolj uporabljenih protokolih na tem področju. Razvili smo štiri vtičnike za platformo OM2M, ki je implementirana v skladu s specifikacijami standarda OneM2M. Ob začetku pisanja tega dela je platforma imela podporo samo za protokol HTTP, zato je predstavljala dober temelj za razvoj naše referenčne arhitekture. Opravljen je bil pregled literature, platform, protokolov in odprtokodnih knjižnic, na podlagi katerih smo implementirali dva splošna vtičnika za protokola CoAP in MQTT ter dva vzorčna vtičnika za tehnologiji Z-Wave in ZigBee. Cilj naloge je bil uporabiti odprtokodno platformo interneta stvari, ki temelji na določenem aktualnem standardu in jo je mogoče razširiti tako, da bo podpirala celoten nabor protokolov, ter z uporabo odprtokodnih knjižnic implementirati dodatke za komunikacijo po izbranih protokolih. Rezultat naloge je odprtokodna rešitev, ki realizira ključne protokole interneta stvari in katere prosta dostopnost bo gotovo pripomogla k večji interoperabilnosti na tem področju.

Ključne besede: internet stvari, platforme, interoperabilnost, standardi, komunikacijski protokoli.

Abstract

Title: Referential architecture for the development of interoperable solutions in the Internet of Things

This thesis describes a referential architecture of the internet of things, consisting of a hardware and a software part, used for communication of devices, which are using the most widely used protocols of this domain. We have developed four plugins for the OM2M platform, which is implemented in accordance with the specifications of the OneM2M standard. When we started writing, the OM2M platform supported only the HTTP protocol and therefore represented a good basis for the development of our referential architecture. An initial overview of the literature, platforms, protocols and open source libraries was made, based on which we implemented two general bindings for the protocols CoAP and MQTT, and two sample plugins were developed for the technologies Z-Wave and ZigBee. The aim of this thesis was to use an open source platform of internet of things, which would base on a relevant standard, and expand it, using the open source libraries, so it would support a set of relevant protocols and enable their usage. The result of this thesis is an open source solution, which implements the key internet of things protocols and which, by being open source, will surely contribute to the interoperability of devices in this domain.

Keywords: internet of things, platforms, interoperability, standards, communication protocols.

Poglavje 1

Uvod

Koncept interneta stvari (angl. *internet of things*, v nadaljevanju IoT) se v zadnjih letih hitro razvija. Besedna zveza predstavlja skupen izraz za povezovanje različnih tehnologij, naprav, objektov in storitev. Dnevno lahko na spletnih straneh, kot sta na primer kickstarters.com in indiegogo.com, spremljamo pojav velikega števila novih naprav in storitev, s katerimi poskušajo inovatorji in raziskovalci prodreti na to področje ter upajo na uspeh. Internet stvari se je pojavil kot rezultat združevanja različnih tehnoloških razvojnih področij. Natančneje, pojavil se je kot rezultat inovacij na področju identifikacijskih tehnologij (uporaba čipov RFID, črtnih kod) in senzoričnih omrežij. Okoli leta 2000 se je tehnologija RFID začela uporabljati v logistiki za spremljanje dobrin. V enakem obdobju je bilo opravljenih veliko raziskav na področju senzoričnih omrežij in pametnih sistemov. Senzorji so postajali vedno manjši, procesna moč naprav pa je naraščala. Kljub tem napredkom so vse rešitve v tistem obdobju bile razvite za specifične domene, zato ni bilo prave interoprerabilnosti in medpovezljivosti med različnimi aplikacijskimi področji.

V nalogi je naslovljena problematika interoperabilnosti na področju interneta stvari. Poleg velikega števila platform, katerih količina se hitro povečuje, je ključen problem pomanjkanje standardizacije oziroma ignoriranje standardov. Posledično imamo opravka z rešitvami, ki so zaprte in jih je med seboj težko integrirati. Interoperabilnost pa je ključna lastnost in predpogoj za številne scenarije uporabe na področju IoT. Osnovni element večine scenarijev IoT je platforma IoT, ki omogoča komunikacijo med napravami IoT ter zajem in procesiranje podatkov. Potrebujemo platformo, ki bo prosto dostopna (odprtokodna), ki bo temeljila na

uveljavljenih standardih, bo lahka (aplikacijska logika se lahko implementira tudi na mikrokontrolerjih) ter bo podpirala celoten sklad protokolov, ki jih standardi priporočajo.

V nalogi je predstavljen primer takšne rešitve, kjer smo za osnovo vzeli platformo OM2M (temelji na standardu in je v začetku realizirala le komunikacijo prek protokola HTTP) ter jo razširili tako, da podpira celoten sklad protokolov. Te smo v platformo vključili z uporabo odprtokodnih knjižnic, upoštevajoč standard OneM2M. Prosta dostopnost takšne platforme bo gotovo pripomogla k večji interoperabilnosti, kar je naš osnovni cilj.

Magistrsko delo, skupaj z uvodom, sestavlja sedem poglavij. V drugem predstavimo pomembnejša sorodna dela z vidika platform IoT.

V tretjem opišemo pomembnejše protokole IoT ter naredimo kratek pregled številnih konzorcijev in zvez, ki delujejo na področju standardizacije IoT.

Četrto poglavje je namenjeno predstavitvi platforme OM2M. Poglavje začnemo s splošnim opisom platforme, ki mu sledi predstavitev zgradbe sporočil. Nadaljujemo z opisom postopka nameščanja platforme na naprave z operacijskim sistemom Linux, nato pa predstavimo še glavne gradnike platforme - vtičnike IPU. Na koncu predstavimo vse zahtevke, ki so potrebni za registriranje naprave na platformi.

V petem poglavju je predstavljena struktura referenčne arhitekture. V prvem delu je predstavljen strojni del referenčne arhitekture, v drugem pa programski del. V opisu programskega dela je predstavljena struktura posameznega vtičnika za določen protokol, opisano pa je tudi njegovo delovanje.

V šestem poglavju je predstavljen pregled celotnega dela, poleg tega pa opravimo primerjavo z že obstoječimi platformami.

V zadnjem, sedmem, poglavju predstavimo nekaj primerov uporabe naše razširjene platforme.

Poglavje 2

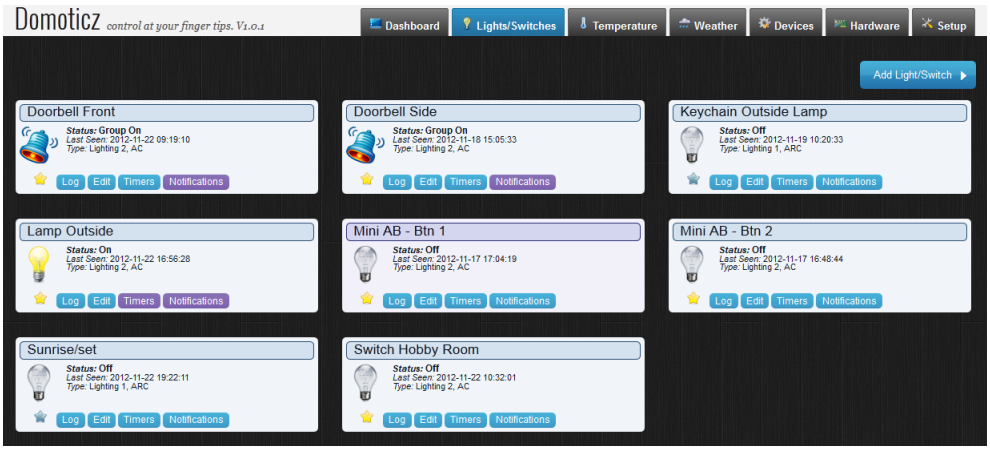
Sorodne platforme

Pri izbiri platforme interneta stvari (v nadaljevanju IoT), ki smo jo uporabili v naši referenčni arhitekturi, smo izhajali iz naslednjih zahtev: platforma je morala biti prosto dostopna, odprtokodna in je morala podpirati vsaj enega od danes ključnih protokolov IoT. Razširili smo pregled platform, ki smo ga opravili že za potrebe prijave naloge, ter zbrali seznam platform, ki je predstavljen v prilogi C. Izbrali smo štiri platforme, ki so se na spletu največ omenjale in izpolnjujejo prej omenjene pogoje, zato s tega stališča predstavljajo sorodna dela. Platforme smo ocenjevali glede na čas, ki je potreben za vključitev določene naprave (testirali smo na vključitvi naprave Z-Wave, saj te za delovanje običajno ne zahtevajo dodatne konfiguracije, ter enega mikrokontrolerja NodeMCU, ki ga je bilo treba v platforme največkrat vključiti v obliki virtualne naprave), in z vidika tehnične zahtevnosti glede na povprečnega uporabnika.

2.0.1 Domoticz

Platforma [34] podpira številne naprave, iz objav na spletu pa je razvidno, da se največ uporablja za upravljanje brezžičnih naprav, ki komunicirajo na frekvenci 433 MHz. Prek storitve Notify My Android omogoča potisna sporočila (angl. *push notifications*) za telefone z operacijskim sistemom Android oziroma za iOS prek storitve Prowl. Pri raziskovanju te platforme je bilo na spletu zaslediti največ kritik glede podpore za mobilne aplikacije (za telefone z operacijskim sistemom Android je bila pred kratkim posodobljena). Platforma ima podporo za zaznavanje pre-

prostih kompleksnih dogodkov (na podlagi tipa naprave), njene funkcionalnosti pa se da razširiti s skriptami v različnih skriptnih in programskih jezikih (PHP, Python, Lua, Bash). Za vključevanje preprostih mikrokontrolerjev (na primer Arduino) se najpogosteje uporabljajo skripte, ki so del projekta MySensors [36]. Prek vključenega grafičnega vmesnika si je mogoče ogledati zgodovino meritev v obliki grafov. Glavni prednosti platforme sta enostavnost uporabe ter preprost grafični vmesnik, ki se ga da poljubno prilagoditi, zato je ta platforma primerna predvsem za ljudi z manj tehničnega znanja. Za namestitev platforme in vključitev ene naprave, ki je komunicirala po protokolu Z-Wave, smo potrebovali petnajst minut. Nekatere podprte naprave in dodatki (RaZberry, PiFace, upravljanje vhodov in izhodov) so na voljo samo, če platformo uporabljamo skupaj z računalnikom Raspberry Pi. Platforma ne podpira protokolov CoAP, Bluetooth, ZigBee. Pri protokolu MQTT omogoča objavljanje obvestil v teme s preddefinirano obliko naslova. Če želimo v platformo vključiti napravo (na primer mikrokontroler NodeMCU), ki ni privzeto podprta, ustvarimo tako imenovano virtualno napravo ter jo upravljamo prek spletnega vmesnika platforme s sporočili v formatu JSON. Za vključitev enega mikrokontrolerja NodeMCU smo potrebovali eno uro. Platforma podpira vključitev naprav, ki komunicirajo po protokolu Z-Wave, prek knjižnice OpenZwave ali prek knjižnice ZwaveMe. Uporabniški vmesnik platforme je prikazan na sliki 2.1.



Slika 2.1: Uporabniški vmesnik platforme Domoticz.

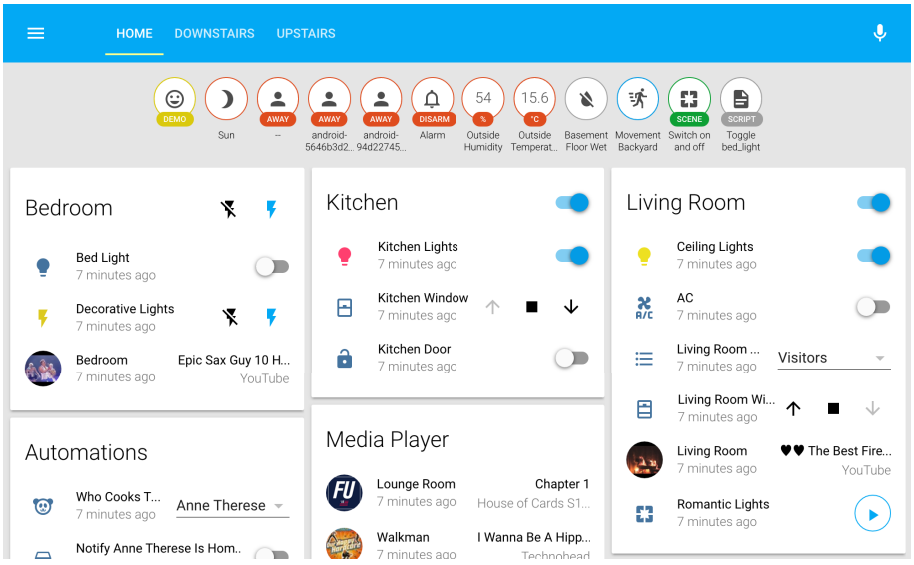
2.0.2 HomeGenie

Platforma HomeGenie [32] ima nekaj zanimivih funkcionalnosti. Podpira izdelovanje scenarijev na podlagi makrov, ki jih posname uporabnik. Vsebuje podporo za protokol Universal plug and play (UPnP) in glasovno upravljanju naprav. Ima vgrajen urejevalnik in prevajalnik za številne skriptne in programske jezike. Platforma privzeto ne podpira protokolov CoAP, ZigBee, Bluetooth, je pa to mogoče vključiti v obliki razširitev, ki jih platforma imenuje avtomatizacijski programi (angl. *automation programs*). Platformo sestavlja več modulov (en proces za vsak modul), ki omogočajo komuniciranje naprav po različnih protokolih. Platforma omogoča podporo za lastne razširitve s programi, napisanimi v jezikih JavaScript, Ruby, Python, C#. Na sistemih Linux omogoča upravljanje naprav z infrardečim daljinskim upravljalnikom. Za namestitev platforme in vključitev ene naprave, ki je komunicirala po protokolu Z-Wave, smo potrebovali dvajset minut. Za vključitev mikrokontrolerja NodeMCU smo potrebovali eno uro. Slika 2.2 prikazuje grafični vmesnik platforme.

Slika 2.2: Uporabniški vmesnik platforme HomeGenie.

2.0.3 HomeAssistant

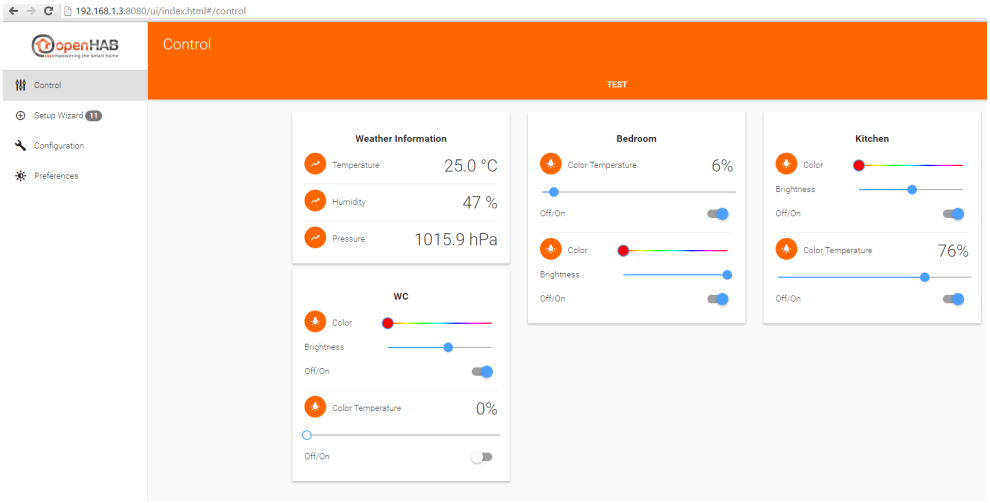
Odprtokodna platforma HomeAssistant [33] vsebuje podporo za številne komercialne izdelke, kot sta na primer luč Philips Hue in termostat Nest. Določene komercialne naprave lahko zazna samodejno, brez dodatne konfiguracije. Platforma podpira računalnik Raspberry Pi in je v celoti napisana v skriptnem jeziku Python 3 (na sistemih Linux sta za namestitev potrebni dve vrstici). Od vseh platform vključuje največ podpore za številne priljubljene storitve, kot so na primer IFTTT, PushBullet, medijski predvajalnik Kodi, predvajalnik Plex. Platforma omogoča enoten spletni vmesnik, ki je enak za osebne računalnike in mobilne naprave. Na voljo so številni vtičniki za različna orodja, storitve, tehnologije. Eden od bolj zanimivih vtičnikov je vtičnik za orodje Nmap, s katerim lahko ugotovi, ali je nekdo doma (na podlagi statičnega naslova ali imena gostitelja). Platforma omogoča definiranje tako imenovanih stikal, ki uporabniku omogočajo izvajanje konzolnih ukazov. Podpira upravljanje naprav ob proženju dogodkov s pomočjo tako imenovanih avtomatizacijskih pravil. Ob namestitvi na operacijski sistem Linux se na napravi ustvari skrita mapa, ki vsebuje konfiguracijsko datoteko s končnico yaml. Za namestitev platforme in vključitev ene naprave, ki je komunicirala po protokolu Z-Wave, smo potrebovali pol ure. Za vključitev mikrokontrolerja NodeMCU smo potrebovali pol ure. Vmesnik platforme je predstavljen na sliki 2.3.



Slika 2.3: Uporabniški vmesnik platforme HomeAssistant.

2.0.4 OpenHAB

Platforma [31] je sestavljena iz aplikacij OpenHAB Runtime in OpenHAB Designer. Prva aplikacija poganja številne vtičnike, ki omogočajo komuniciranje naprav po različnih protokolih, druga aplikacija pa predstavlja orodje, s katerim lahko urejamo konfiguracijske datoteke naših naprav. Za lažje izmenjevanje konfiguracijskih datotek platforma ponuja podporo za protokol Samba ali oblačno storitev DropBox. Predpogoj za poganjanje platforme je, da naprava podpira virtualno napravo Java (angl. *Java virtual machine*). Platforma omogoča vnaprej pripravljene grafične vmesnike (HTML5), z njo pa je mogoče komunicirati tudi prek tekstovnih vmesnikov (vmesnik za protokol XMPP za pošiljanje sporočil v Javi, konzola ogrodja OSGi in koledar Google za ročno urejanje dogodkov, za katere želimo, da jih platforma izvrši). Na voljo so tudi mobilne aplikacije za različne platforme (Android, iOS, Windows Phone). Zanimiva funkcionalnost mobilnih aplikacij je branje ploščic NFC, ki jih lahko uporabimo za upravljanje naših naprav (recimo za vklop luči v sobah, kjer ne moremo vgraditi dodatnih stikal v stene). Trajno shranjevanje podatkov, ki jih pozneje lahko vizualiziramo, je podprto v obliki shranjevanja v lokalne podatkovne baze (MySQL, Db40, RRDTool) ali pa s shranjevanjem in vizualiziranjem podatkov z uporabo ogrodij Sense, Xively, ThingsSpeak. Glavna lastnost platforme je njena modularnost (uporablja se ogrodje OSGi), ki spodbuja k ponovni uporabi z vtičniki. Ti so na voljo za različne storitve, protokole, naprave. Tekstovno konfiguriranje platforme poteka v datoteki *items* (deklaracija naprav, konfiguracija funkcionalnosti, uporabljenih protokolov in storitev), datoteki *sitemap* (urejanje vmesnika spletne aplikacije platforme), datoteki *rules* (določanje logike naprav in avtomatizacijskih pravil, ki določajo obnašanje naprav pri določenih pogojih). Sintaksa konfiguracijskih datotek je napisana v gramatiki Xtext DSL. Za namestitev platforme in vključitev ene naprave, ki je komunicirala po protokolu Z-Wave, smo potrebovali dve uri. Za vključitev mikrokontrolerja NodeMCU smo potrebovali eno uro. Vključitev naprav v platformo je bila veliko bolj težavna in zamudna v primerjavi s preostalimi platformami, saj je za vključitev treba sestaviti ustrezne konfiguracijske datototeke.



Slika 2.4: Uporabniški vmesnik platforme OpenHAB.

Poglavje 3

Standardi na področju IoT

Z besedo *standard* običajno označujemo metode, norme in regulacije, na podlagi katerih morajo biti razviti produkti in storitve. Standardi so lahko uradni in obvezujoči (*de jure*), lahko pa so neuradni (*de facto*). Zadnje definirajo podjetja ali skupina podjetij (interesne skupine, konzorciji, zveze, združenja), ki se želijo najprej uveljaviti na določenem področju. Standardi imajo veliko vlogo pri uveljavljanju novih tehnologij, ker omogočajo različnim deležnikom uporabo enakih in povezljivih sistemov [10]. Še posebej je to pomembno v globalnih sistemih, kjer njihova kompleksnost zahteva uporabo platform, ki so implementirane v skladu s standardi. Po drugi strani so lahko standardi v določenih primerih tudi škodljivi, saj njihovo uveljavljanje zahteva veliko časa, poleg tega pa se v realnosti pojavljajo tudi primeri, kjer mora proizvajalec izbirati med implementacijo produkta po določenem standardu ali tako, da bo kompatibilen z vsemi (ali vsaj večino) izdelki, ki so trenutno na trgu in teh standardov ne upoštevajo. Čakanje na uveljavitev standarda lahko povzroči izgubo številnih poslovnih priložnosti. Kljub temu se v splošnem standardizacijske procese obravnava kot pozitivno stvar. Številne organizacije na področju interneta stvari se ukvarjajo s standardi, uradnimi in neuradnimi, ter različnimi aplikacijskimi področji. Številna podjetja uporabljajo lastne rešitve ter se zanašajo na uporabo lastniških ali odprtih protokolov, vmesnikov in platform, ki lahko sobivajo v sistemih. Iz dozdajšnjega razvoja je že jasno, da bodo sistemi IoT uporabljali tako uradne standarde kot tudi nekatere neuradne, ki jih bodo potrdile različne zveze ali podjetja v vodilnih vlogah, zato bo ključno sodelovanje različnih organizacij, ki delujejo na tem področju.

3.0.1 Konzorciji, društva in zveze

Zaradi naraščajoče uporabe brezžičnih komunikacijskih tehnologij in potrebe po novih uporabniških izkušnjah narašča tudi potreba po konsistentnih standardih in široki interoperabilnosti, ki bi IoT spremenila v monoliten, povezan ekosistem. V zadnjih letih so se oblikovale številne zveze in konzorciji [10, 56] z namenom izpolnjevanja novih zahtev na tem področju. Njihova naloga je zagotoviti standardiziran način komuniciranja in deljenja informacij med vsemi deležniki ter zagotavljanje platforme, kjer zadnji lahko preverijo pravilnost implementacije svoje rešitve v skladu z izbranim standardom. Te rešitve so nato ponujene za uporabo tudi preostalim deležnikom. Standardizacija je potrebna za maksimalno fleksibilnost in skalabilnost IoT, za zagotavljanje izboljšav na področju funkcionalnosti, stroškov in kakovosti med različnimi aplikacijami in rešitvami ter za zagotavljanje krajšega razvojnega cikla, ki podjetjem omogoča hitrejše dostavljanje rešitev na trg. Za zagotavljanje popolne povezljivosti med stvarmi (angl. *things*) morajo standardi definirati vmesnike na tehnični in aplikacijski ravni. Proces standardizacije na področju interneta stvari otežujejo tudi proizvajalci naprav, ki v svoje produkte vključujejo tehnologije, ki jih še niso potrdila uradna standardizacijska telesa [9].

Slika 3.1 prikazuje organizacije, ki delujejo na različnih področjih standardizacije IoT, in njihovo delitev. V glavnem lahko te organizacije razdelimo v tri večje skupine.

Prva skupina organizacij, "Organizacije, osredotočene na tehnološko arhitekturo rešitev", se osredotoča na razvoj tehnološke arhitekture rešitev. Organizacije v tej skupini lahko naprej delimo v tri podskupine, ki se razlikujejo glede na to, za kateri sloj definirajo rešitve. V prvo podskupino spadajo organizacije, ki definirajo standarde na fizični ravni. Njihov namen je definiranje specifikacij za zagotavljanje interoperabilnosti med napravami in storitvami ter spodbujanje k uveljavitvi izbranih tehnologij na trgu. Med bolj poznane organizacije v tej skupini spadajo organizacija IEEE (tehnologija Ethernet), ZigBee alliance (tehnologija ZigBee) in WiFi alliance (tehnologija Wi-Fi). V drugo skupino spadajo organizacije, ki definirajo standarde na sejni ravni, transportni ravni, na področju sporočilnih protokolov ali na področju semantike. Najbolj poznana organizacija na sliki, ki spada v to skupino, je organizacija IETF, saj je med drugim zaslužna za definiranje

protokola HTTP, protokola CoAP, 6LowPAN ter številnih drugih. Organizacije v tretji podskupini "Večnivojske rešitve" se ukvarjajo z zagotavljanjem standardov, ki definirajo rešitve na več ravneh (za razliko od predhodnjih dveh skupin). Razvijalcem zagotavljajo odprtokodne platforme, ki poenostavijo odkrivanje sorodnih naprav, njihovo medsebojno komuniciranje z uporabo standardnih vmesnikov in definiranje novih vmesnikov. Tako kot pri preostalih skupinah je tudi cilj teh organizacij zagotavljanje interoperabilnosti med napravami, storitvami in aplikacijami ter definiranje certifikacijskih programov. V to skupino spada tudi organizacija OneM2M, ki je predstavljena v tej nalogi ter definira standard OneM2M.

Drugo večjo skupino, "Vertikalno osredotočene organizacije", sestavljajo organizacije, ki proizvajalcem naprav omogočajo grajenje interoperabilnih rešitev v industriji na izbranih področjih (zdravstvo, transport, pametni domovi, logistika ter drugih). Zavzemajo se za določitev certifikacijskih postopkov, za potrjevanje interoperabilnega delovanja rešitev na določenih področjih in za sodelovanje z vladnimi regulativnimi agencijami, za zagotavljanje metod za varno in učinkovito upravljanje raznolikih rešitev. Primer dveh organizacij iz te skupine sta organizacija Continua, ki deluje predvsem na področju zdravstva, ter organizacija Zwave Alliance. Ta je zadolžena za certificiranje naprav Z-Wave, ki se najpogosteje uporabljajo v hišni avtomatizaciji.

Namen organizacij iz tretje skupine, "Marketing in izobraževanje", je zagotavljanje poslovnih priložnosti svojim članom v obliki spoznavanja, povezovanja in sodelovanja s podjetji, strankami, dobavitelji, razvijalci in potencialnimi investitorji na področju IoT. Ponujajo sodelovanje v delovnih skupinah in možnost sooblikovanja novih standardov ter odpravljanje pomanjkljivosti pri starejših. S pomočjo spletnih strani, namenskih dogodkov in tiskovnih konferenc promovirajo in obveščajo svoje člane o novih produktih in pomembnejših spremembah.

Organizacije se razlikujejo tudi glede na to, kdo se lahko včlani v njih in kakšen je znesek članarine. Na eni strani so organizacije, katerih člani so lahko posamezniki in imajo brezplačno članstvo, na drugi strani pa imamo organizacije, v katere se lahko včlanijo le podjetja, znesek članarine pa znaša nekaj tisoč evrov.

Slika 3.1: Pregled zvez in konzorcijev interneta stvari.

3.0.2 Protokoli

Internet je močno spremenil način komuniciranja in sodelovanja med ljudmi, saj se je z njegovim razvojem razširil dostop do brezplačnih informacij, kar je posledično preoblikovalo naša življenja tako, kot si je bilo v preteklosti težko predstavljati. IoT ne daje poudarka na komunikacijo med ljudmi, temveč so v ospredju pametne, povezane naprave, ki morajo biti sposobne med seboj komunicirati z večjimi hitrostmi in sposobnostmi, kot so potrebne za ljudi. Podatke, ki jih zbirajo, morajo shranjevati ter prepošiljati na ustrezno strežniško infrastrukturo. Ta mora z uporabo različnih protokolov zagotavljati podatke preostalim napravam, programom za analiziranje in ljudem.

Na voljo so številni protokoli, ki se med seboj razlikujejo po arhitekturi, velikosti sporočil, hitrosti, funckionalnostih in po količini pomnilnika, ki je potrebna za njihovo delovanje na napravah. Izbor protokolov, ki smo jih vključili v platformo OM2M, ni bil naključen. Na sliki 3.2 so predstavljene štiri večje organizacije, ki delujejo na področju IoT, in njihove lastnosti, razdeljene po izbranih ravneh referenčnega modela OSI.

	OIC	AllSeen	Thread	oneM2M
Člani	Intel, Samsung, CISCO, GE Software, ter 71 drugih	Qualcomm, LG, Microsoft, Sony, Panasonic, ter 151 drugih	Nest (Google), Frees cale, ARM, Samsung, Silicon Labs, ter 134 drugih	ARIB, ATIS, CCSA, ETSI, TIA, TSDSI, TTA, TTC, ter 222 ostalih
Aplikacijski nivo	HTTP, CoAP	Bus Attachment, AllJoyn, Message, Routing Node, AllJoyn Transport	-	HTTP, CoAP, MQTT
Transportni nivo	TCP/UDP	TCP/UDP	TCP/UDP	TCP/UDP
Omrežni nivo	IPv4/IPv6	IPv4/IPv6	6LoWPAN (IPv6)	IPv4/IPv6(6LoWPAN)
Podatkovni nivo	-			
Fizični nivo	(Bluetooth, WiFi, Zigbee, Z-Wave, ANT, 3G/4G, ter ostale tehnologije)			

Slika 3.2: Pregled zvez in konzorcijev interneta stvari.

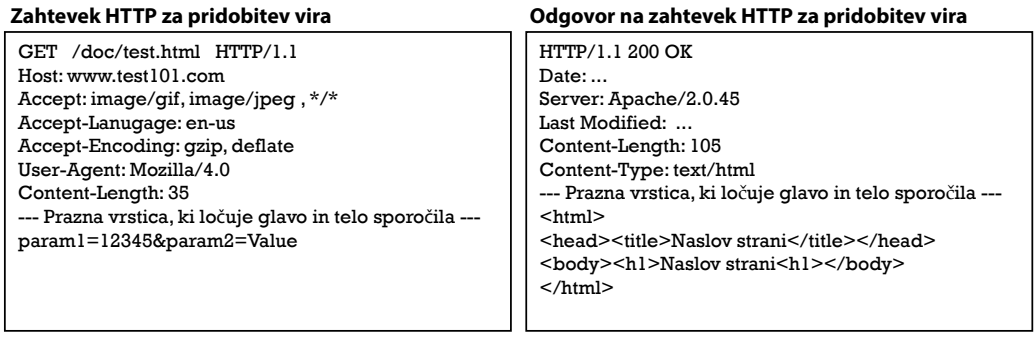
Poleg pomembnejših članov posameznih organizacij, uporabljenih paketov na

transportni ravni, uporabljenih protokolih na omrežni ravni in podprtih tehnologijah na fizični ravni so na sliki predstavljeni tudi podprti protokoli na aplikacijski ravni in podprte tehnologije na fizični ravni. Standard OneM2M, s katerim smo se ukvarjali v okviru naloge, za zdaj na aplikacijski ravni definira specifikacije za protokole HTTP, CoAP in MQTT, zato smo v okviru naloge implementirali splošna vtičnika za protokola CoAP in MQTT (vtičnik za protkol HTTP je privzeto že vključen v platformo). S slike je razvidno, prek katerih tehnologij lahko poteka komunikacija na fizični ravni. Zaradi dostopa do potrebne strojne opreme in naprav smo se odločili za implementacijo vzorčnih vtičnikov za tehnologiji Zigbee in Z-Wave. Odkrili smo, da protokol Bluetooth nima ustrezne knjižnice, ki bi jo lahko uporabili pri izdelavi vtičnika. Na voljo je sicer knjižnica Bluecove, vendar smo pri prebiranju objav na spletu zasledili, da se knjižnica ne posodablja zaradi težav z licenciranjem, poleg tega pa obstoječa različica ne deluje na računalniku Raspberry Pi 3.

V nadaljevanju so predstavljeni protokoli, za katere so bili izdelani vtičniki za platformo OM2M. Pri protokolih HTTP, CoAP in MQTT, za katere obstajajo specifikacije za splošne vtičnike in je potrebno poznavanje strukture zahtevkov zaradi preslikovanja polj med sporočili izbranega protokola in sporočilom platforme OM2M, smo opisali tudi strukturo zahtevkov. Standard OneM2M ne definira specifikacij za naprave, ki komunicirajo po protokolu Z-Wave ali ZigBee, zato v poglavjih, ki obravnavajo ta dva protokola, nismo opisali strukture sporočil.

3.0.2.1 HTTP

Protokol HTTP je danes najpomembnejši protokol, ki se uporablja na internetu, saj je potreben za delovanje spletnih strani, spletnih storitev in delovanje fizičnih naprav, ki so povezane v internet. Glavna komponenta protokola je zahtevek HTTP, ki se deli na zahtevek za pridobitev spletnega vira (angl. *HTTP request*) in odgovor na zahtevek za pridobitev spletnega vira (angl. *HTTP response*). Poznavanje strukture zahtevkov pri protokolu HTTP in naslednjih dveh protokolih je potrebno zaradi implementacije splošnih vtičnikov za platformo OM2M, ki upoštevajo specifikacije standarda OneM2M [4, 5, 6, 7, 8].



Slika 3.3: Struktura sporočila protokola HTTP.

Slika 3.3 prikazuje strukturo obeh zahtevkov. Oba tipa sta sestavljena iz glave in telesa. V prvi vrstici zahtevka za pridobitev vira se nahajajo polja, ki nam povedo, nad katerim virom se bo izvedla izbrana operacija ter katera različica protokola HTTP je v uporabi. Pri odgovoru na zahtevek za pridobitev vira v tej vrstici najdemo različico protokola, ki se uporablja, ter dve polji, ki nam sporočata, kakšen je bil rezultat izvedene operacije. Prvo polje vsebuje rezultat v obliki števila, namen druge pa je ubesediti rezultat operacije. Po prvi vrstici se v glavi zahtevkov nahajajo različna polja, ki vsebujejo informacije o pošiljatelju in prejemniku sporočila ter njene metapodatke. Za glavo se v obeh zahtevkih nahaja prazna vrstica, ki ločuje glavo in telo sporočila. Na koncu zahtevkov najdemo v obeh zahtevkih telo sporočila. Pri zahtevku za pridobitev vira najdemo v telesu vsebino sporočila, ki jo sestavljajo številni parametri, medtem ko se pri odgovoru na zahtevek na tem mestu nahaja vsebina vira (če zahtevamo stran, napisano v označevalnem jeziku HTML, bomo kot odgovor prejeli izvorno kodo te strani). Preslikovanje posameznih polj zahtevkov HTTP v ustrezne dele sporočil platforme OM2M je podrobneje definirano v specifikacijah standarda OneM2M za protokol HTTP.

3.0.2.2 CoAP

Protokol CoAP [12] je v marsičem podoben protokolu HTTP. Naprave, ki komunicirajo po tem protokolu, so nastavljive prek naslovov IP, njihove storitve pa lahko prožimo prek enoličnih naslovov, tako kot pri storitvah REST. Oba protokola imata nekaj skupnih metod (GET, POST, PUT, DELETE) ter definirata podobne

šifrante za odzive na zahtevke (404, 500). Ker specifikacije standarda OneM2M definirajo več kod za odzive na zahtevke, kot jih definira protokol CoAP, se več kod standarda OneM2M preslika v enako kodo protokola CoAP. Preslikave so definirane v tabeli 6.2.4-1 [6]. Glavna razlika med protokoloma (poleg tega, da HTTP uporablja protokol TCP, COAP pa protokol UDP) je v tem, da je CoAP namenjen napravam, ki imajo na voljo malo sistemskih virov (procesorske moči, delovnega spomina).

Paketi protokola so sestavljeni iz številnih neobveznih polj. Če ta polja niso prisotna, se velikost paketa občutno zmanjša, zaradi česar so primernejši za omenjene naprave. Slika 3.6 prikazuje strukturo sporočila CoAP in njegovo delitev na obvezna polja, ki so prikazana osiveno, in neobvezna polja. Sporočila protokola so omejena z maksimalno velikostjo paketa UDP. Sestavljena so iz glave, polja "Token", neobveznih polj ter telesa.

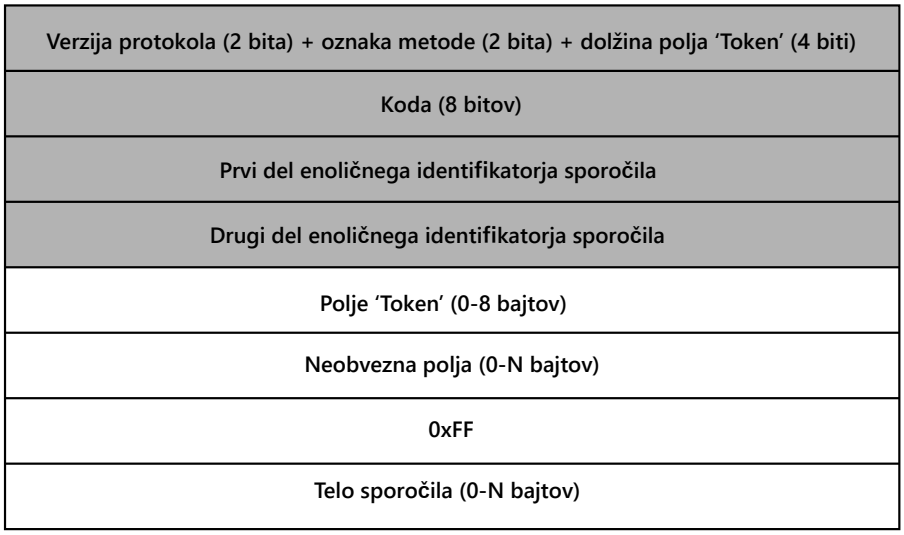
Pomen vsakega polja je naslednji:

- **Glava** - Maksimalna velikost je štiri bajte. Vsebuje različico protokola, tip sporočila (CON, NON, RST, ACK), dolžino polja "Token", vrsto zahtevka (GET, POST, PUT, DELETE) ter enolični identifikator sporočila.

- **Polje Token** - Polje, ki se uporablja za korelacijo sporočil (če ta potujejo v različnem vrstnem redu, kot so bila poslana). To polje je lahko dolgo od nič do osem bajtov.

- **Neobvezna polja** - Neobvezna polja so po namenu ekvivalentna atributom v glavi zahtevka HTTP. Vsako neobvezno polje enolično določa trimestno število. Mogoče je dodati lastna neobvezna polja, kot smo naredili pri implementaciji vtičnika za ta protokol, kar je opisano v poglavju 5.2.2 .

- **Telo** - Vsebina sporočila, ki se lahko prenaša v različnih formatih (XML, JSON, različni formati standarda OneM2M).



Slika 3.4: Struktura sporočila protokola CoAP.

Za protokol je na voljo več odjemalcev, najpopularnejša pa sta Copper [52] in Libcoap [53]. Copper je na voljo kot vtičnik za brskalnik Firefox, zgrajen je na knjižnici Californium, nima pa podpore za večje število neobveznih polj (angl. *options*). V vmesniku vtičnika so v obliki drevesne strukture predstavljeni vsi viri naprave, nad katerimi je mogoče izvajati določene operacije. Poleg tega vmesnik vsebuje številna polja, s katerimi je mogoče nastavljati vrednosti neobveznih polj. Libcoap je za razliko od odjemalca Copper konzolna aplikacija, ki je po funkcionalnostih zelo primerljiva prej omenjenemu vtičniku. Postopek ročne namestitve je prikazan na sliki 3.1.

```
1  sudo apt-get update
2  sudo apt-get install dh-autoreconf
3  wget http://downloads.sourceforge.net/project/libcoap/coap-18/libcoap-4.0.3.tar.gz
4  tar -xzf libcoap-4.0.3.tar.gz
5  rm libcoap-4.0.3.tar.gz
6  cd libcoap-4.0.3
7  autoconf
8  ./configure
9  make
10 sudo cp examples/coap-client /usr/local/bin/
```

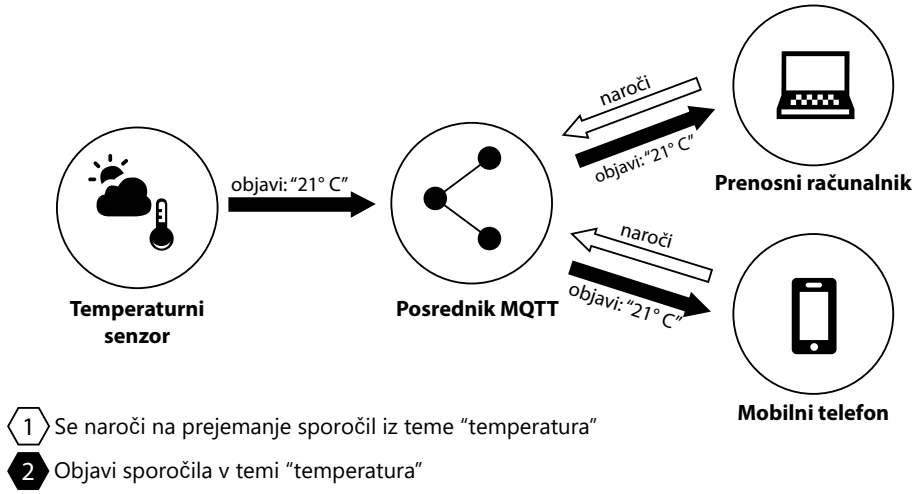
Izvorna koda 3.1: Postopek ročne namestitve odjemalca Libcoap

Aplikacija definira mnogo stikal, s katerimi lahko natančneje določimo njeno delovanje. S stikalom "-b" lahko na primer definiramo velikost bloka, če je sporočilo pred prenosom treba fragmentirati. Ko je odjemalec nameščen, lahko z naslednjo vrstico pošljemo zahtevo, s katero pridobimo vrednost nekega vira:

```
1 coap-client -m Get coap://192.168.0.110:5683/pot_do_vira
```

3.0.2.3 MQTT

Protokol MQTT (včasih je ta beseda predstavljala kratico, zdaj pa ne več), ki ga je leta 1999 razvilo podjetje IBM, je sporočilni protokol z arhitekturo objavi-naroči (angl. *publish-subscribe*), ki je prikazana na sliki 3.5. Sestavlja jo posrednik (angl. *broker*) ter eden ali več odjemalcev, ki lahko pošiljajo sporočila ali pa se naročijo na njihovo prejemanje s prijavitvijo v izbrane teme (angl. *topic*).

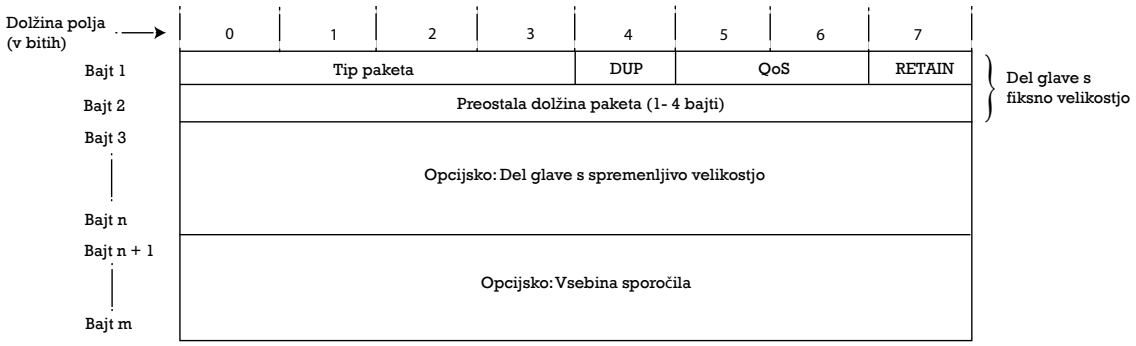


Slika 3.5: Arhitektura objavi-naroči.

Protokol je preprost, odprtokoden ter zasnovan tako, da ga je enostavno implementirati. V zadnjih letih se je pojavilo veliko število knjižnic, ki implementirajo

strežniški del protokola ali odjemalca. Najbolj razširjeni knjižnici za implementacijo strežniškega dela protokola sta Mosquitto [23] in Moquette [24], za implementiranje odjemalca pa je na voljo knjižnica Eclipse Paho [22].

Paket protokola MQTT sestavljajo trije deli: del glave s fiksno velikostjo, drugi del glave s spremenljivo velikostjo ter telo sporočila.



Slika 3.6: Struktura sporočila protokola MQTT.

Del glave s fiksno velikostjo vsebuje informacije o tipu paketa, različne zastavice (pri paketu PUBLISH so to zastavice DUP, QoS in RETAIN) ter polje, v katerem je navedena preostala dolžina paketa. Tipi paketov so predstavljeni s štirimi biti, katerih vrednosti prikazuje tabela 3.1. Zastavice omogočajo nastavljanje različnih možnosti prenosa in hrambe sporočil in so odvisne od tipa paketa. Teoretična velikost paketa je 256 MB, vendar je dobra praksa, da se podatke z večjo velikostjo fragmentira v pakete z velikostjo do 100 kB ter pošlje v obliki paketov tipa PUBLISH.

Protokol ima nekaj posebnosti:

- Definira različne stopnje zanesljivosti dostave sporočila (angl. *Quality of service*), ki so opisane v nadaljevanju.
- Omogoča shranjevanje sej vsakega odjemalca na posredniku.
- Če odjemalec nepričakovano izgubi povezavo s posrednikom, lahko zadnji o tem obvesti vse druge odjemalce, ki so naročeni na sporočila določene teme (angl. *Last will and testament*).

- Podpora vzorcem (regularnim izrazom) v definiranju imen tem, na katere se odjemalci naročajo. Ta možnost omogoča odjemalcem, da se lahko hkrati prijavijo v več tem.
- Če odjemalec izgubi povezavo s posrednikom, se bo poskušal ponovno povezati nanj samodejno. Posrednik lahko ima v tem trenutku še delno odprto povezavo do odjemalca. Ko se odjemalec ponovno poveže, se ta seja uniči, nato pa se ustvari nova z enakim enoličnim identifikatorjem odjemalca (angl. *client take-over*).

Pošiljatelj in prejemnik sporočila lahko skupaj definirata parameter, ki določa zanesljivost dostave sporočila (angl. *Quality of service*). Parameter ima lahko tri vrednosti:

- **0** - Pošiljatelj pošlje sporočilo prejemniku, ki ne shrani sporočila in ne potrdi njegovega prejema (angl. *fire and forget*).
- **1** - Pošiljatelj pošilja sporočilo, dokler od prejemnika ne prejme potrditve, da je sporočilo prejel (angl. *at least once*). V tem načinu lahko prejemnik večkrat prejme sporočilo (pošiljatelj še ni prejel potrditve).
- **2** - Ta način omogoča, da prejemnik prejme sporočilo samo enkrat. Pošiljatelj najprej objavi sporočilo, ki ga prejemnik potrdi. Ko pošiljatelj prejme potrditev, lahko izbriše poslano sporočilo, ker ve, da ga je prejemnik prejel. Nato si pošiljatelj in prejemnik izmenjata dodatne pakete, s katerimi pobrišeta obstoječe shranjeno stanje.

Oznaka	**Vrednost**	**Opis**
RESERVED	0	Ni definirano.
CONNECT	1	Paket odjemalca za povezavo na strežnik.
CONNACK	2	Paket za potrjevanje paketa z oznako CONNECT.
PUBLISH	3	Paket za objavljanje sporočil.
PUBACK	4	Paket za potrjevanje paketa PUBLISH.

PUBREC	5	Paket za potrjevanje prejema paketa PUBLISH (pri QOS=2).
PUBREL	6	Paket, ki ga odjemalec pošlje kot odgovor na posrednikov paket PUBREC, ali paket, ki ga posrednik pošlje kot odgovor na odjemalčev paket PUBREC.
PUBCOMP	7	Paket, ki ga posrednik pošlje kot odgovor na odjemalčev paket PUBREL, ali paket, ki ga odjemalec pošlje kot odgovor na posrednikov paket PUBREL.
SUBSCRIBE	8	Paket odjemalca, s katerim se naroči na prejemanje sporočil določene teme.
SUBACK	9	Paket za potrjevanje paketa SUBACK.
UNSUBSCRIBE	10	Paket odjemalca, s katerim se odjavi od prejema sporočil določene teme.
UNSUBACK	11	Paket za potrjevanje paketa UNSUBACK.
PINGREQ	12	Paket, s katerim odjemalec preveri delovanje posrednika.
PINGRESP	13	Paket, ki ga posrednik pošlje kot odgovor na paket PINGREQ.
DISCONNECT	14	Paket odjemalca, s katerim se odjavi s posrednika.
RESERVED	15	Ni definirano.

Tabela 3.1: Paketi protokola MQTT.

3.0.2.4 Zwave

Protokol Z-Wave je leta 1990 razvilo dansko podjetje ZenSys. Začetna ideja snovalcev protokola razviti lasten sistem hišne avtomatizacije je kmalu prerasla v komunikacijsko tehnologijo s poudarkom na interoperabilnosti med napravami. Tehnologija se je najprej razširila na ameriške trge (predvsem zaradi že prisotnih tehnologij na področju hišne avtomatizacije), sledila ji je Evropa, nazadnje pa se je pojavila tudi na azijskih trgih. Podjetje ZenSys je pozneje kupilo ameriško podjetje Sigma Designs, ki danes skrbi za razvoj protokola (definira komunikacijo na fizični ravni) ter je edini proizvajalec čipov Z-Wave. Leta 2005 je bila ustanovljena fundacija Z-Wave Alliance, ki združuje proizvajalce naprav, skrbi za marketinške dogodke in spodbuja interoperabilnost med napravami s procesom certificiranja. Ko naprava uspešno prestane certificiranje, je zabeležena v bazo [48], proizvajalec pa lahko nanjo prilepi posebno nalepko, s katero potrjuje, da je naprava prestala certificiranje.

Protokol je na začetku veljal za zaprtega, s katerim so lahko delali le partnerji omenjenega podjetja, pozneje pa so se pojavile številne odprtokodne implementacije predvsem zaradi vse večjega uspeha protokola. Za najbolj razširjeno odprtokodno implementacijo velja knjižnica OpenZwave, ki je implementirana v programskih jezikih C++, C#, C ter Python. Protokol omogoča postavitev robustnih zankastih omrežij (angl. *mesh networks).* Glavni element vsakega omrežja Z-wave je upravljalec (angl. *controller*), ki se uporablja za vključevanje naprav Z-Wave v omrežje. Upravljalce lahko najdemo v različnih oblikah; lahko so samostojne enote, ki poleg protokola Z-Wave podpirajo še druge protokole, najpogosteje pa jih zasledimo v obliki ključev USB [40, 41, 42, 43], ki so prikazani na sliki 3.7. Ključ AEON GEN5, na sliki predstavljen s številko 1, je novejša različica ključa AEON Series 2, na sliki označen s številko 5. Oba ključa se od drugih upravljalcev USB razlikujeta v tem, da podpirata ročno vključevanje in izključevanje naprav iz omrežja prek namenskega gumba. Glavna razlika med ključema je ta, da ključ GEN5 podpira tudi novejše naprave, naprave Z-Wave Plus, in se lahko uporablja kot orodje za diagnosticiranje stanja omrežja na podlagi vgrajene barvno kodirane diode. Številki 3 in 4 označujeta enaka ključa, RaZberry, ki se razlikujeta v načinu priklopa. Drugega prodajajo kot dodatek za računalnik Raspberry Pi in ga priklopimo na splošno namenske pine. Ta ključ se od drugih loči tudi v tem, da je treba

za uporabo vseh njegovih funkcionalnosti dokupiti posebno licenco. S številko 2 je označen upravljalec, ki poleg protokola Z-Wave podpira tudi druge protokole (Wi-Fi, Bluetooth).

Slika 3.7: Različni upravljalci protokola Z-Wave.

Običajen postopek vključitve naprave Z-wave sestavljata dva koraka. Uporabnik najprej postavi upravljalca v stanje za vključitev (angl. *inclusion mode*). Omenjeno stanje predstavlja časovni interval, med katerim bo naprava poslušala za zahteve po vključitve v omrežje. Ko je upravljalec v stanju za vključitev, uporabnik pritisne še gumb za vključitev v omrežje na napravi Z-wave ali pa jo samo priklopi na napajanje, če naprava podpira samodejno vključevanje (angl. *auto-inclusion mode*). Vsako omrežje je enolično določeno z atributom `HOME_ID` (4 bajti), vsako napravo pa enolično določa atribut `NODE_ID` (1 bajt), ki se ob uspešni vključitvi v omrežje shrani na upravljalca. Njihove funkcionalnosti določajo razredi (angl. *COMMAND_CLASSES*), ki jih implementirajo razvijalci. Seznam

razredov in njihov pomen je predstavljen v prilogi A.

Naprave lahko podpirajo več razredov, obvezno pa morajo podpirati razred `BASIC`, ki omogoča upravljanje in pridobivanje stanja osnovne funkcionalnosti naprave. Vsak razred je sestavljen iz ene ali več metod, ki jih lahko pošljemo napravi. Pri razredu `BASIC` je to metoda `BASIC_SET`, ki omogoča upravljanje osnovne funkcionalnosti naprave. V primeru žarnice bi osnovno funcionalnost predstavljalo njeno prižiganje in ugašanje. Pridobivanje stanja naprave omogoča metoda `BASIC_GET`, `BASIC_REPORT` pa je metoda, s katero naprava odgovori na metodo `BASIC_GET` in vsebuje trenutno stanje naprave.

3.0.2.5 ZigBee

ZigBee je še ena tehnologija, ki omogoča izdelavo zankastih omrežij. Za razliko od tehnologije Z-Wave, ki deluje na frekvenčnem območju 900 MHz, naprave, ki implementirajo to tehnologijo, delujejo v frekvenčnem območju 2.4 GHz. Pogosto enačimo poimenovanji ZigBee in XBee, kar ni pravilno, saj prvo označuje standardni komunikacijski protokol, drugo pa najbolj popularno znamko modulov, ki lahko komunicirajo po različnih protokolih (ZigBee, 802.15.4, Wi-Fi). Module XBee v glavnem delimo glede na serijo; modul je lahko serije 1 ali serije 2. Moduli obeh serij so na voljo v dveh različicah, ki imata različno oddajno moč. Moduli, ki imajo običajno oddajno moč, se preprosto imenujejo XBee, medtem ko se moduli z večjo oddajno močjo imenujejo XBee PRO. Moduli različnih serij so med seboj nekompatibilni, zato mora vsako omrežje uporabljati le module določene serije. Vsi moduli XBee delujejo z voltažami do 3.3 V.

Tehnologija definira tri vloge vozlišč:

- **Koordinator** - Vozlišče, ki ima glavno vlogo v omrežju. Njegove naloge so grajenje omrežja, razdeljevanje naslovov, vzdrževanje varnosti in stanja omrežja.

- **Usmerjevalnik** - Vozlišče, ki se lahko pridruži drugim omrežjem, posreduje sporočila drugih vozlišč, pošilja svoja sporočila ter usmerja sporočila v omrežju. Ta vozlišča morajo biti nujno stalno priklopljena na napajanje, zato da zagotavljajo konstantno in zanesljivo posredovanje sporočil.

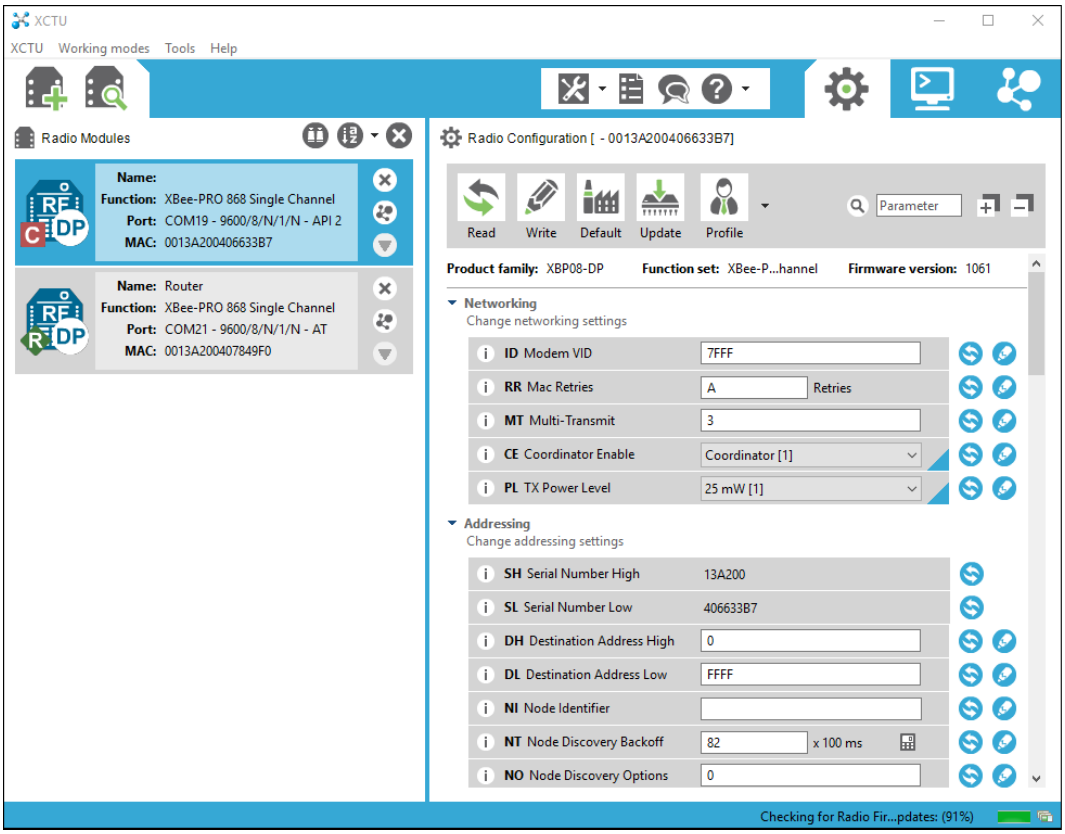
- **Končno vozlišče** - Vozlišča se lahko pridružijo omrežjem in lahko pošiljajo ali prejemajo sporočila. Glavna razlika med to vrsto vozlišča in usmerjevalnim vozliščem je, da ne posredujejo sporočil drugih naprav, zato ni potrebe, da imajo stalno napajanje.

Najmanjše omrežje, ki uporablja tehnologijo ZigBee, je tako sestavljeno iz para koordinator - usmerjevalnik oziroma koordinator - končno vozlišče. Modulom se da spreminjati vlogo z ustreznim orodjem. V okviru naloge smo se ukvarjali z znamko modulov XBee, ki jih lahko konfiguriramo z orodjem X-CTU. Orodje, katerega grafični vmesnik je prikazan na sliki 3.8, poleg nastavljanja vloge vozlišč omogoča tudi spreminjanje različnih parametrov vozlišča, kot na primer: moč oddajanja, način delovanja, naslov prejemnika sporočil, možnosti enkripcije, nastavljanje stanj digitalnih in analognih izhodov ter drugih parametrov. Omogoča tudi nadgrajevanje programske opreme modulov in ima vgrajeno konzolo, prek katere lahko testiramo delovanje komunikacije. Orodje je na voljo na uradni strani proizvajalca modulov, podjetja Digi [47]. Moduli niso združljivi z razvojnimi ploščami (angl. *breadboard*), saj imajo manjši razmik nožic, kot jih imajo plošče. Če želimo modul uporabljati na razvojni plošči, je treba priskrbeti ustrezen adapter.

Modulom lahko definiramo različne načine delovanja. V glavnem lahko modul deluje v načinu AT ali v načinu API. Način AT se naprej deli na transparentni način (angl. *transparent mode*) in ukazni način (angl. *command mode*). V transparentnem načinu modul vse, kar prejme, v nespremenjeni obliki pošlje naprej. Ukazni način se uporablja za konfiguriranje modula z uporabo ukazov AT. Z izbiro načina delovanja definiramo, kako bomo modul konfigurirali (posredno definiramo serijsko komunikacijo). Pomanjkljivost načina AT je, da ne omogoča pošiljanja stanj in upravljanja vhodov modula. V ta namen se uporablja način API. Ko modul enkrat deluje v načinu API, ga lahko konfiguriramo samo z orodjem X-CTU [13].

Za konfiguriranje modulov poleg omenjene aplikacije potrebujemo tudi ustrezno strojno opremo. Če konfiguriramo module XBee, imamo na voljo dve možnosti. Uporabimo lahko namensko razvojno ploščo, ki je namenjena hitremu prototipiranju in vključuje nekaj dodatnih stvari, ki olajšajo razvoj (gumbi za reset modula, gumbi za spreminjanje stanj vhodov, različne indikatorske luči LED). Preizkusili smo plošči z oznakama XBIB-U-DEV (vmesnik USB) in XBIB-R-DEV (vmesnik RS-232), s katerima je mogoče programirati vse vrste modulov XBee razen sta-

rejših modulov tipa S6. Cena posamezne razvojne plošče je okoli 50 evrov. Če bi radi samo konfigurirali modul in ne potrebujemo raznih dodatkov, ki jih ima razvojna plošča, lahko kupimo namenski adapter za module, ki je cenejši (okoli 20 evrov) in ga lahko vstavimo tudi v razvojno ploščo.

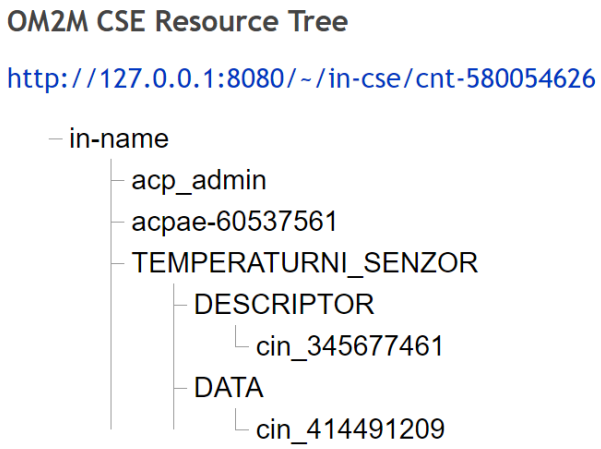


Slika 3.8: Orodje X-CTU

Poglavje 4

Platforma OM2M

Platforma OM2M [15, 1, 2, 3] je delo francoskega laboratorija LAAS-CNRS. Na voljo je v dveh različicah. Prva, različica 0.8, implementira standard ETSI Smart M2M, medtem ko novejša različica, 1.0, implementira standard OneM2M. Registrirane naprave so predstavljene v obliki drevesne strukture virov, ki jih lahko spreminjamo z operacijami storitev REST. Slika 4.1 prikazuje videz drevesne strukture virov enote IN-CSE, ko je na platformi registrirana ena naprava z imenom TEMPERATURNI_SENSOR.



Slika 4.1: Prikaz naprave na platformi OM2M.

Obe različici platforme sta si, čeprav upoštevata različna standarda, zelo podobni. Standard ETSI velja za predhodnika standarda OneM2M, zato v obeh

različicah platforme najdemo številne enake koncepte. Implementaciji obeh sestavljata množici vtičnikov, ki so ločeni po funkcionalnosti. Tako nova kot tudi stara platforma omogočata vnaprej pripravljena splošna vtičnika za protokola HTTP in CoAP, za preostale protokole pa je treba izdelati namenske vtičnike, ki preslikajo sporočilo določenega protokola v neodvisno obliko, ki jo platforma razume. V tej nalogi smo se osredotočili na uporabo novejše različice, ki temelji na standardu OneM2M. Standard lahko predstavimo z modelom funkcionalne arhitekture sistema OneM2M [4]. Ta model je namenjen predstavitvi pomembnejših entitet, ki sestavljajo arhitekturo sistema OneM2M. Pomembno je poudariti tudi, da so te entitete logični objekti, za katere ni nujno, da predstavljajo fizične naprave.

1. **Aplikacijska entiteta** (angl. *Application entity - AE*): Aplikacijska entiteta je entiteta aplikacijske ravni, ki predstavlja aplikacijsko logiko določene entitete sistema M2M. Izvajajoče instance so enolično določene z identifikatorjem AE-ID (v primeru platforme OM2M se ti identifikatorji posredujejo ob začetni registraciji naprave). Primer aplikacijske entitete bi bila aplikacijska logika za merjenje količine sladkorja v krvi ali aplikacijska logika za merjenje električne porabe energije v hišah.

2. **Entiteta standardnih storitev** (angl. *Common service entity - CSE*) zagotavlja nabor osnovnih storitev sistemov M2M. Te entitete so na voljo drugim entitetam skozi referenčne točke (Mca, Mcc). Vsaka entiteta standardnih storitev je določena z identifikatorjem CSE-ID (pri platformi OM2M je zapisan v konfiguracijski datoteki config.ini, ki jo najdemo v mapi configuration, pri vozliščih IN-CSE in MN-CSE). Entiteta omogoča naslednje standardne storitve: registracija naprav, odkrivanje naprav, upravljanje skupin, lokacijske storitve, upravljanje naprav, zagotavljanje varnosti, upravljanje naročanj na obvestila.

3. **Entiteta omrežnih storitev** (angl. *Network service entity - NSE*): Ta entiteta omogoča nižje ležeče storitve omrežja entitetam standardnih storitev. Primeri takšnih storitev so upravljanje naprav, proženje naprav, prenos podatkov med različnimi entitetami sistema OneM2M.

4. **Referenčna točka Mca** označuje komunikacijsko povezavo med aplikacijsko entiteto in entiteto standardnih storitev. Ta povezava omogoča aplika-

cijskim entitetam uporabo storitev entitete standardnih storitev, zadnjim pa zagotavlja možnost komunikacije z aplikacijskim entitetami. Aplikacijska entiteta se v sistemu OneM2M lahko nahaja v različnih fizičnih entitetah.

5. **Referenčna točka Mcn** označuje komunikacijsko povezavo med entiteto standardnih storitev in entiteto omrežnih storitev. Ta povezava entiteti standardnih storitev omogoča uporabo storitev entitete omrežnih storitev.

Vozlišča so logične entitete, ki so individualno nastavljive v sistemih OneM2M. Standard definira več tipov vozlišč, za potrebe te naloge pa je dovolj, če opišemo naslednje:

1. **Končno vozlišče** (angl. *Application dedicated node - ADN*): vozlišče vsebuje vsaj eno entiteto AE ter nobene entitete CSE. V sistemu OneM2M je lahko več takšnih vozlišč.

2. **Vmesno vozlišče** (angl. *Middle node - MN*): vozlišče vsebuje eno entiteto CSE ter eno ali več entitet AE. V sistemu OneM2M je lahko več takšnih vozlišč.

3. **Glavno vozlišče** (angl. *Infrastructure node - IN*): vozlišče vsebuje eno entiteto CSE ter eno ali več entitet AE. V sistemu OneM2M je lahko natančno eno tako vozlišče. Komunikacija s preostalimi entitetami CSE in AE poteka prek referečnih točk Mca in Mcn.

Obstaja mnogo konfiguracij in načinov medsebojnega povezovanja različnih entitet sistema OneM2M. V primeru naše arhitekture smo izbrali trinivojsko arhitekturo, ki je sestavljena iz več instanc aplikacijskih entitet (to so lahko fizične naprave, na katere so povezani senzorji ali aktuatorji), ene ali več instanc entitete MN-CSE, ter ene entitete IN-CSE.

4.1 Opis platforme

V platformo OM2M lahko vključimo tri tipe naprav:

1. Naprava, ki ne komunicira po protokolu IP (na primer naprave, ki komunicirajo po protokolu Z-Wave).

2. Starejša (angl. *legacy*) naprava, ki komunicira po protokolu IP ter ne upošteva specifikacij standarda OneM2M (primer bi bila naprava, ki na določenem naslovu v formatu JSON omogoča vrednost temperaturnega senzorja).

3. Naprava, ki komunicira po protokolu IP v skladu s specifikacijami standarda OneM2M.

Za prvi in drugi tip naprave mora v našem primeru platforma poskrbeti za ustvarjanje ustrezne strukture na platformi: registracija naprave, izdelava vsebnikov za opis in meritve naprave (za protokol HTTP smo ta postopek opisali v poglavju 4.5). Naprave, omenjene v drugi točki, predstavljajo predkonfigurirane naprave, ki jim ni mogoče spreminjati aplikacijske logike. Za pridobivanje vrednosti je treba spisati vtičnik IPU, ki bo z nje pridobival meritve. Za naprave, omenjene v tretji točki, je treba izdelati vtičnik IPU (po specifikaciji standarda OneM2M: CoAP, MQTT), ki predvideva, da bo naprava samo poslala vse ustrezne zahtevke za registracijo in opis naprave ter nato pošiljala svoje meritve. S to točko smo se ukvarjali, ko smo za protokola CoAP in MQTT izdelali splošna vtičnika, ki vključujeta osnovne operacije, omenjene že v splošnem opisu platforme. Ta vtičnik predstavlja splošno obliko vtičnika za določen protokol. Specifikacije med drugim definirajo način registracije naprave, preslikovanje sporočil OM2M v sporočilo določenega protokola ter njihovo izmenjevanje.

Pri platformi OM2M se vse mertive določenega senzorja nahajajo znotraj vsebnika `DATA`. Poleg tega platforma OM2M omogoča dodatne filtre, s katerimi lahko pridobimo samo podmnožico meritev (angl. *discovery*).

Naprava se lahko naroči na prejemanje obvestil drugih naprav. Ta mehanizem je pri platformi OM2M implementiran v obliki naročanja na spremembe pri določenih napravah (angl. *subscriptions*).

4.2 Zgradba sporočil

Platforma OM2M definira lasten format sporočil [4] za izvajanje določenih operacij. Sestavljena so iz obveznih in neobveznih polj, določa jih tip operacije, ki se prenaša v sporočilu.

Vsak zahtevek mora vsebovati naslednja polja:

- **To** - Naslov vira, prejemnika sporočila.
- **From** - Naslov vira, pošiljatelja sporočila.
- **Operation** - Polje z vrsto operacije, ki se bo izvedla na platformi. Mogoče operacije so: Create, Retrieve, Update, Delete, Notify.

Opcijsko lahko zahtevki vsebujejo naslednja polja:

- **Content** - Ta del zahtevka je prisoten samo, če ustvarjamo nov vir ali ga posodabljamo.
- **Role** - Parameter, ki se uporablja za nadzorovanje dostopa do virov platforme na podlagi uporabniških vlog (administrator, navaden uporabnik).
- **Name** - Določa ime vira, ki ga želimo ustvariti.
- **Originating timestamp** - Časovna značka, ki označuje čas ustvarjanja zahtevka.
- **Request expiration timestamp** - Parameter označuje čas veljavnosti zahtevka za pridobitev nekega vira. Če se takšen zahtevek ne nahaja na ustrezni enoti CSE, ga bo trenutna enota poskušala dostaviti na ustrezno enoto CSE, dokler mu veljavnost ne poteče.
- **Result expiration timestamp** - Čas veljavnosti rezultata zahtevka za pridobitev nekega vira.
- **Response type** - Parameter določa tip zahtevka, ki se bo poslal v odgovor na zahtevek za pridobitev nekega vira. Poleg tipa določa, kdaj se bo odgovor poslal pošiljatelju zahtevka za pridobitev vira.
- **Result content** - Določa vsebino odgovora na zahtevek, ki je odvisna od parametra Operation.
- **Result persistance** - Določa časovni interval, v katerem bo na voljo rezultat zahtevka.
- **Request identifier** - Enolični identifikator sporočila.

- **Operation execution time** - Določa čas izvedbe operacije v zahtevku na izbrani enoti CSE.
- **Event category** - Določa kategorijo dogodka, ki se bo uporabila za izvedbo zahtevka.
- **Group Request Identifier** - Določa enolični identifikator skupine zahtevkov.
- **Filter criteria** - Nabor parametrov, s katerimi lahko dodatno omejimo iskani rezultat.

Nekateri protokoli (na primer protokol MQTT) prenašajo med vozlišči celotna sporočila v serializirani obliki (formatu XML ali JSON). Slika 4.1 prikazuje serializirano sporočilo v formatu XML.

```
<m2m:rqp xmlns:m2m="http://www.onem2m.org/xml/protocols">
    <op>1</op>
    <to>~/in-cse</to>
    <fr>admin:admin</fr>
    <ty>2</ty>
    <nm>TEMPERATURNI_SENZOR</nm>
    <pc> <!-- vsebina telesa, ki ustreza shemam standarda OneM2M -->
        <om2m:ae xmlns:om2m="http://www.onem2m.org/xml/protocols">
            <api>app-sensor</api>
            <lbl>Type/sensor Category/temperature Location/home</lbl>
            <rr>false</rr>
        </om2m:ae>
    </pc>
</m2m:rqp>;
```

Izvorna koda 4.1: Primer serializiranega sporočila.

Serializacija in deserializacija sporočil v platformi OM2M sta implementirani v projektu z imenom /org.eclipse.om2m.datamapping.jaxb (metodi objToString(Object obj) in stringToObj(String representation) razreda Mapper.java). Kakor je razvidno že iz imena, se za serializacijo in deserializacijo sporočil uporablja knjižnica JAXB (angl. *Java Architecture for XML Binding*), ki omogoča preslikovanje med

različnimi sporočilnimi formati in objekti programskega jezika Java. Na sliki 4.2 sta prikazani vrstici, ki poskrbita za serializacijo in deserializacijo sporočil.

```
1 // Serializacija
2 String representation_XML = DataMapperSelector.getDataMapperList().get(
      MimeMediaType.XML).objToString(request);
3
4 // Deserializacija
5 RequestPrimitive primRequest = (RequestPrimitive) DataMapperSelector.
      getDataMapperList().get(MimeMediaType.XML).stringToObj(
      secondSerializedRequest);
```

Izvorna koda 4.2: Postopek serializacije in deserializacije sporočil.

RequestPrimitive in ResponsePrimitive sta imeni objektov, ki v implementaciji platforme označujeta sporočila standarda OneM2M.

4.3 Namestitev, prevajanje in konfiguracija

Postopek nameščanja, prevajanja in konfiguriranja platforme za naprave, na katerih teče operacijski sistem Windows, je podrobno opisan v vodičih, ki jih najdemo na uradni strani platforme [45]. V tem poglavju želimo opisati ta postopek še za naprave, na katerih teče operacijski sistem Linux (platformo smo poskusili namestiti na računalnika Raspberry Pi 3 in Raspberry Pi B+). Platforma za delovanje potrebuje Javo (vsaj 1.7) in orodje Maven (vsaj verzijo 3) za prevajanje in upravljanje knjižnic, ki so potrebne za njeno delovanje. Na naši napravi najprej izvedemo naslednja dva ukaza:

```
1 apt-get install update
2 apt-get install upgrade
```

S prvim ukazom pridobimo informacije o posodobitvah paketov, ki so že nameščeni

na napravi, z drugim ukazom pa te posodobitve prenesemo na napravo. Ko so paketi posodobljeni, lahko preverimo, ali že imamo nameščeno Javo in aplikacijo Maven. To lahko storimo z naslednjima dvema ukazoma, ki ju izvedemo v terminalu:

```
mvn --version
java -version
```

Če je na napravi že nameščena aplikacija Maven, se bo v terminalu izpisalo naslednje:

```
Apache Maven 3.3.9
(bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T16:41:47+00:00)
Maven home: /opt/apache-maven-3.3.9
Java version: 1.8.0_65, vendor: Oracle Corporation
Java home: /usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt/jre
Default locale: en_GB, platform encoding: UTF-8
OS name: "linux", version: "4.1.19-v7+", arch: "arm", family: "unix"
```

Podobno se bo po izvedbi drugega ukaza izpisalo za Javo (v zadnjih različicah operacijskega sistema Raspbian je Java že prednameščena):

```
java version "1.7.0_101"
OpenJDK Runtime Environment (IcedTea 2.6.6) (7u101-2.6.6-2~deb8u1+rpi1)
OpenJDK Zero VM (build 24.95-b01, interpreted mode)
```

Ko sta ti dve aplikaciji nameščeni, lahko namestimo še orodje git. To orodje predstavlja sistem za verzioniranje kode in ga bomo uporabili zato, da nam ne bo treba ročno prenašati izvorne kode platforme iz njenega repozitorija. Orodje namestimo z naslednjim ukazom:

```
apt-get install git
```

Preverimo še, ali je bilo uspešno nameščeno:

```
git --version
```

Zdaj imamo na napravi nameščeno vse, kar potrebujemo za delovanje platforme. Z naslednjim ukazom prenesemo izvorno kodo platforme iz njenega repozitorija:

```
git clone https://git.eclipse.org/r/om2m/org.eclipse.om2m
```

Po prenosu se pomaknemo v glavno mapo projekta ter prevedemo izvorno kodo:

```
cd org.eclipse.om2m/
mvn clean install
```

Prevajanje izvorne kode na računalniku Raspberry Pi 3 traja štiri minute. Na starejših različicah računalnika Raspberry Pi se med procesom prevajanja izvorne kode lahko javi izjema OutOfMemmoryError, ki se pojavi zaradi pomanjkanja pomnilnika (Raspberry Pi B+ ima 512 MB pomnilnika, Raspberry Pi 3 pa dvakrat več). To napako lahko odpravimo z naslednjim ukazom, s katerim lahko spremenimo količino dodeljenega pomnilnika:

```
java -Xms<zacetna velikost sklada> -Xmx<maksimalna velikost sklada>
```

Če je bila izvorna koda platforme uspešno prevedena, dobimo kot rezultat tri mape, ki ustrezajo enotam IN-CSE, MN-CSE, ASN-CSE. Znotraj vsake mape najdemo datoteko start.sh, s katero zaženemo posamezno enoto, in config.ini, ki vsebuje konfiguracijske podatke enote.

V našem primeru smo želeli uporabiti trinivojsko arhitekturo, ki bo sestavljena iz enote IN-CSE (osebni računalnik z naslovom IP 192.168.0.101), ene enote MN-CSE (računalnik Raspberry Pi z naslovom IP 192.168.0.202) in naprav, na katere

bodo povezani senzorji ali aktuatorji (na primer mikrokontroler NodeMCU s senzorjem temperature in vlage). Za konfiguriranje enote IN-CSE sledimo vodiču na uradni strani platforme za naprave, na katerih teče operacijski sistem Windows. Naš računalnik Raspberry Pi bo imel vlogo enote MN-CSE, zato ga je treba ustrezno konfigurirati. Na napravi se pomaknemo v mapo, ki ustreza enoti MN-CSE, ter poiščemo njeno konfiguracijsko datoteko. V datoteki ustrezno spremenimo ali samo preverimo pravilnost vrednosti naslednjih lastnosti:

```
1 org.eclipse.om2m.cseBaseId=mn-cse
2 org.eclipse.om2m.cseBaseName=mn-name
3 org.eclipse.om2m.cseBaseAddress=192.168.0.202
4 org.eclipse.equinox.http.jetty.http.port=8282
5 org.eclipse.om2m.remoteCseId=in-cse
6 org.eclipse.om2m.remoteCseName=in-name
7 org.eclipse.om2m.remoteCseAddress=192.168.0.101
8 org.eclipse.om2m.remoteCsePort=8080
```

Lastnost cseBaseAddress je treba nastaviti na naslov naprave, ki predstavlja enoto MN-CSE, zato da bo mogoče dostopati do drevesne strukture enote prek spletnega vmesnika. Če želimo imeti več enot MN-CSE, postopek ponovimo. Če se nam enota MN-CSE prikaže v spletnem vmesniku platforme, ne moremo pa dostopati do njenega drevesnega vira, lahko najprej preverimo, ali delamo z zadnjo različico platforme (mogoče je tudi, da nam težave povzročajo nastavitve požarnega zidu).

4.4 Vtičniki IPU

Vključevanje naprav, ki komunicirajo po različnih protokolih, je omogočeno v obliki vtičnikov IPU (angl. *Interworking Proxy Unit*) [46], ki predstavljajo osrednji gradnik platforme. Pri raziskovanju smo zasledili, da jih lahko najdemo v dveh oblikah. V prvem primeru imamo splošni vtičnik IPU. Ta pretvori sporočilo določenega protokola, ki upošteva specifikacije standarda OneM2M, v ustrezno sporočilo standarda OneM2M. Druga oblika vtičnikov je namenjena specifičnim napravam, za katere ni na voljo specifikacija standarda OneM2M oziroma jih želimo vključiti v

platformo na svoj način.

Prvo obliko vtičnikov običajno sestavlja razred za strežniški del komunikacije (sprejem, pretvorba in odgovor na prejet zahtevek v določenem protokolu), razred, ki implementira odjemalca, ter pomožni razredi.

Drugo obliko vtičnikov sestavlja pet razredov:

1. **Activator** - razred, namenjen aktiviranju in deaktiviranju vtičnika.
2. **Monitor** - razred, katerega naloge so ustvarjanje ustreznih vsebnikov novih naprav, poslušanje za nove zahtevke ter posredovanje zahtevka.
3. **OBixUtil** - naprave so v platformi OM2M privzeto predstavljene v formatu oBIX [39], ki predstavlja standardizirano obliko sporočil XML za opisovanje stavb. Razred omogoča dinamično grajenje opisov naprav v tem formatu.
4. **Controller** - razred, ki na podlagi prejetega zahtevka izvede želeno operacijo ter njen rezultat prikaže v spletnem vmesniku platforme.
5. **RequestSender** - pomožni razred, ki olajša gradnjo sporočil platforme OM2M.

Vzorčni primer takšnega vtičnika je predstavljen na strani platforme [45].

4.5 Prijava naprav

Prijavljanje naprav na platformo poteka v obliki zahtevkov določenih protokolov, ki enkapsulirajo operacije platforme OM2M. Ko platforma prejme te zahtevke, jih pretvori v lasten format in izvede. Po izvedbi operacij se rezultat sporoči pošiljatelju z obratnim postopkom pretvorbe sporočil, na platformi pa si lahko uporabnik ogleda nabor vsebnikov in njihovih instanc, s katerimi je naprava predstavljena. Naprave se na platformo lahko prijavljajo same, mogoče pa jih je registrirati tudi na druge načine. Lahko jih na primer ustvarimo ročno z uporabo odjemalcev za določen protokol ali pa sporočila za njihovo ustvarjanje vključimo v druge naprave, ki bodo namesto njih poslale ustrezne zahtevke. V nadaljevanju je predstavljen primer registracije naprave, imenovane TEMPERATURNI_SENZOR, in izdelave njenih vsebnikov z zahtevki protokola HTTP.

4.5.1 Ustvarjanje glavnega vira

Prvi zahtevek je namenjen izdelavi glavnega vira, ki bo predstavljal napravo in bo vseboval vse druge vsebnike in njihove instance. Na naslov izbranega vozlišča IN-CSE pošljemo zahtevek, ki ima v glavi definiranih nekaj dodatnih polj. Ta polja so definirana v specifikaciji standarda OneM2M protokola HTTP [7]. Polje X-M2M-Origin je namenjeno avtentikaciji pošiljatelja. V polju Content-type je zapisan format telesa sporočila. Polje ty (angl. *type*) vsebuje tip vira, ki bo ustvarjen na platformi ob izvedbi tega zahtevka. V tem primeru želimo ustvariti vir tipa AE, s katerim lahko predstavimo napravo. Seznam vseh podprtih virov si lahko ogledamo v tabeli 6.3.4.2.11-1 [5]. Zadnje polje, X-M2M-NM, vsebuje ime naprave, ki jo želimo registrirati. V telesu sporočila se nahaja dokument XML, ki vsebuje dodatne podatke o napravi, ki jo želimo registrirati. Pravilnost dokumentov se preverja s shemami XML, ki jih definira standard OneM2M in so vključene v platformo OM2M.

Polje	Vrednost
URL	http://127.0.0.1:8080/~/in-cse
Metoda	POST
Polja v glavi zahtevka	X-M2M-Origin: admin:admin Content-Type: application/xml;ty=2
Telo zahtevka	<om2m:ae xmlns:om2m="http://www.onem2m.org/xml/protocols" rn="TEMPERATURNI_SENZOR"> <api>app-sensor</api> <lbl>Tip/senzor Kategorija/temperatura Lokacija/spalnica</lbl> <rr>true</rr> </om2m:ae>

Slika 4.2: Ustvarjanje glavnega vsebnika

4.5.2 Ustvarjanje vsebnika DESCRIPTOR

Z drugim zahtevkom ustvarimo vsebnik `DESCRIPTOR`, katerega namen je enkapsulacija instanc, ki bodo hranile metapodatke o napravi. Polje ty v tem zahtevku označuje ustvarjanje vsebnika. Naslov zahtevka zdaj kaže na unikaten naslov glavnega vira naprave, ki smo ga ustvarili s prejšnjim zahtevkom.

Polje	Vrednost
URL	http://127.0.0.1:8080/~/in-cse/in-name/TEMPERATURNI_SENZOR
Metoda	POST
Polja v glavi zahtevka	X-M2M-Origin: admin:admin Content-Type: application/xml;ty=3
Telo zahtevka	<om2m:cnt xmlns:om2m="http://www.onem2m.org/xml/protocols" rn="DESCRIPTOR"> </om2m:cnt>

Slika 4.3: Ustvarjanje vsebnika DESCRIPTOR.

4.5.3 Ustvarjanje instance vsebnika DESCRIPTOR

S tem zahtevkom ustvarimo instanco vsebnika `DESCRIPTOR`, ki bo hranila metapodatke o napravi in gumbe, s katerimi prožimo operacije, na katere izbrana naprava reagira. Ker ustvarimo instanco vsebnika `DESCRIPTOR`, kaže naslov URL na naslov vsebnika. Polje ty je nastavljeno na vrednost, ki označuje, da ustvarimo instanco nekega vsebnika. Razlaga atributov dokumenta XML v telesu sporočila je podrobneje opisana v tabeli 8.2.3-1 [5].

Polje	Vrednost
URL	http://127.0.0.1:8080/~/in-cse/in-name/TEMPERATURNI_SENZOR/DESCRIPTOR
Metoda	POST
Polja v glavi zahtevka	X-M2M-Origin: admin:admin Content-Type: application/xml;ty=4
Telo zahtevka	<om2m:cin xmlns:om2m="http://www.onem2m.org/xml/protocols"> <cnf>message</cnf> <con> <obj> <str name="tip" val="Temperaturni_senzor"/> <str name="lokacija" val="Spalnica"/> <str name="appId" val="TEMPERATURNI_SENZOR"/> <op name="Pridobi vrednost" href="/in-cse/TEMPERATURNI_SENZOR/DATA/la" in="obix:Nil" out="obix:Nil" is="retrieve"/> </obj> </con> </om2m:cin>

Slika 4.4: Ustvarjanje instance vsebnika DESCRIPTOR.

4.5.4 Ustvarjanje vsebnika DATA

Vsebnik `DATA` vsebuje instance meritev, ki jih naprava pošilja na platformo. Zahtevek je po strukturi enak zahtevku za ustvarjanje vsebnika `DESCRIPTOR`.

Polje	Vrednost
URL	http://127.0.0.1:8080/~/in-cse/in-name/TEMPERATURNI_SENZOR
Metoda	POST
Polja v glavi zahtevka	X-M2M-Origin: admin:admin Content-Type: application/xml;ty=3
Telo sporočila	`<om2m:cnt xmlns:om2m="http://www.onem2m.org/xml/protocols" rn="DATA">` `</om2m:cnt>`

Slika 4.5: Ustvarjanje vsebnika DATA.

4.5.5 Ustvarjanje instance vsebnika DATA

S tem zahtevkom so opisane meritve naprav. Zahtevek je po strukturi enak zahtevku za ustvarjanje instance vsebnika `DESCRIPTOR`. V telesu sporočila so poleg vrednosti meritve navedeni še drugi podatki (enota meritve, tip senzorja, identifikator vtičnika IPU, ki mu je zahtevek namenjen), ki dodatno opisujejo instanco.

Polje	Vrednost
URL	http://127.0.0.1:8080/~/in-cse/in-name/TEMPERATURNI_SENZOR/DATA
Metoda	POST
Polja v glavi zahtevka	X-M2M-Origin: admin:admin Content-Type: application/xml;ty=4

Telo zahtevka	`<om2m:cin xmlns:om2m="http://www.onem2m.org/xml/protocols">`
	`<cnf>message</cnf>`
	`<con>`
	`<obj>`
	`<str name="appId" val="TEMPERATURNI_SENZOR"/>`
	`<str name="kategorija" val="temperatura "/>`
	`<int name="vrednost" val="27"/>`
	`<int name="enota" val="celzija"/>`
	`</obj>`
	`</con>`
	`</om2m:cin>`

Slika 4.6: Instanca vsebnika DATA.

4.5.6 Naročanje na obvestila določene naprave

Če bi bili radi obveščeni o vseh dogodkih nekatere naprave (izvedenih meritvah, proženih akcijah), se lahko na platformi prijavimo na prejemanje njenih obvestil, ki predstavljajo asinhrone dogodke. Polje ty v glavi zahtevka vsebuje vrednost 23, ki predstavlja vir tipa obvestilo. Telo sporočila vsebuje dokument XML, ki definira dva pomembnejša elementa. Prvi določa naslov naprave, ki želi prejemati obvestila, drugi pa format vsebine obvestila.

Polje	Vrednost
URL	http://127.0.0.1:8080/~/in-cse/in-name/TEMPERATURNI_SENZOR/DATA
Metoda	POST
Polja v glavi zahtevka	X-M2M-Origin: admin:admin Content-Type: application/xml;ty=23
Telo zahtevka	<m2m:sub xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="OBVESTILO_TEMPERATURNI_SENZOR"> <nu>http://localhost:1400/monitor</nu> <nct>2</nct> </m2m:sub>

Slika 4.7: Naročanje na obvestila naprave.

4.5.7 Pridobivanje podatkov o napravah

Na podlagi metapodatkov, ki jih zapišemo v instanco vsebnika `DESCRIPTOR`, lahko izvajamo iskanje naprav. Če se na platformi nahaja iskana naprava, v odgovoru prejmemo njen naslov URL. V naslovu URL se nahaja parameter fu (angl. *filter usage*), ki predstavlja filtrirni pogoj. Poleg tega parametra obstajajo še številni drugi, ki so navedeni v tabeli 7.3.3.17-1 [5].

Polje	Vrednost
URL	http://127.0.0.1:8080/~/in-cse?fu=1&lbl=tip/Temperaturni_senzor
Metoda	GET
Polja v glavi zahtevka	X-M2M-Origin: admin:admin
Telo zahtevka	(brez)

Slika 4.8: Naročanje na obvestila naprave.

4.6 Samodejno preusmerjanje zahtevkov

Naprave, na katere so povezani senzorji, lahko registriramo s pošiljanjem točno določenih zahtevkov v ustreznem vrstnem redu, kot je opisano v poglavju 4.5. Podobno lahko storimo tudi za naprave, na katere so povezani aktuatorji. Platforma OM2M omogoča preusmerjanje zahtevkov od vtičnika, ki prejme zahtevek, k naslovljeni napravi (angl. *retargeting*) [5]. V nadaljevanju je predstavljen primer registracije naprave, ki komunicira po protokolu HTTP in na katero je povezan aktuator, ki ga lahko upravljamo prek naslovov /true oziroma /false. Naprava komunicira po protokolu HTTP ter se nahaja na naslovu http://127.0.0.1:1400. Primeri so podani v programskem jeziku Java.

Enako kot pri registraciji naprave s senzorjem je prvi zahtevek namenjen registraciji naprave z izbranim imenom na platformo. V telesu sporočila se pošlje dokument XML, ki ustreza shemam standarda OneM2M ter vsebuje element “poa”(angl. *point of access*). Ko določen vtičnik IPU prejme zahtevek, najprej preveri, ali se v strukturi naprave nahaja omenjeni element. Če se, se zahtevek neodvisno od protokola preusmeri na vneseni naslov, ki ga določa element poa v telesu zahtevka.

```
1  String url = "http://127.0.0.1:8080/~/in-cse";
2
3  HttpClient client = HttpClientBuilder.create().build();
4  HttpPost request = new HttpPost(url);
5
6  request.addHeader("X-M2M-Origin", "admin:admin");
7  request.addHeader("Content-Type", "application/xml;ty=2");
8
9  String content =
10 "<om2m:ae xmlns:om2m=\"http://www.onem2m.org/xml/protocols\"" +
11     "rn=\"TEMPERATURNI_SENZOR\">" +
12   "<api>app-sensor</api>" +
13   "<lbl>Type/sensor Category/temperature Location/home</lbl>" +
14   "<rr>true</rr>" +
15   "<poa>http://127.0.0.1:1400/app</poa>" +
16 "</om2m:ae>";
17
18 HttpEntity entity = new ByteArrayEntity(
```

```
19   content.getBytes("UTF-8"));
20 request.setEntity(entity);
21 HttpResponse response = client.execute(request);
```

Izvorna koda 4.3: Registracija naprave na platformo.

Drugi zahtevek je namenjen ustvarjanju vsebnika `DESCRIPTOR`, ki vsebuje metapodatke naprave v obliki enega podelementa.

```
1  String url = "http://127.0.0.1:8080/~/in-cse/in-name/TEMPERATURNI_SENZOR";
2
3  HttpClient client = HttpClientBuilder.create().build();
4  HttpPost request = new HttpPost(url);
5
6  request.addHeader("X-M2M-Origin", "admin:admin");
7  request.addHeader("Content-Type", "application/xml;ty=3");
8
9  String content = "<om2m:cnt xmlns:om2m=\"http://www.onem2m.org/xml/protocols
       \" rn=\"DESCRIPTOR\"></om2m:cnt>";
10
11 HttpEntity entity = new ByteArrayEntity(
12   content.getBytes("UTF-8"));
13 request.setEntity(entity);
14
15 HttpResponse response = client.execute(request);
```

Izvorna koda 4.4: Ustvarjanje vsebnika DESCRIPTOR.

Tretji zahtevek ustvari instanco vsebnika `DESCRIPTOR`. Z njo poleg metapodatkov o napravi definiramo naslove URL, ki se bodo pripeli vrednosti elementa poa. V primeru, ki je prikazan spodaj, smo definirali dve metodi: prva (z naslovom true) je namenjena prižiganju naprave, druga (z naslovom false) je namenjena ugašanju naprave. Preden vtičnik posreduje zahtevek napravi, sestavi celoten naslov, ki bo aktiviral določeno metodo. V našem primeru sta to naslov "http://127.0.0.1:1400/app?op=true" in naslov "http://127.0.0.1:1400/app?op=false".

```
1  String url = "http://127.0.0.1:8080/~/in-cse/in-name/TEMPERATURNI_SENZOR/
       DESCRIPTOR";
```

```
2
3  HttpClient client = HttpClientBuilder.create().build();
4  HttpPost request = new HttpPost(url);
5
6  request.addHeader("X-M2M-Origin", "admin:admin");
7  request.addHeader("Content-Type", "application/xml;ty=4");
8
9  String content =
10 "<om2m:cin xmlns:om2m=\"http://www.onem2m.org/xml/protocols\">" + "<cnf>
       message</cnf>" +
11   "<con>"
12   + "&lt;obj&gt;&lt;str name=&quot;type&quot; val=&quot;Temperature_Sensor&
       quot;/&gt;&lt;str name=&quot;location&quot; val=&quot;Home&quot;/&gt;&lt;str
        name=&quot;appId&quot; val=&quot;TEMPERATURNI_SENZOR&quot;/&gt;&
       lt;op name=&quot;Turn on&quot; href=&quot;/in-cse/in-name/
       TEMPERATURNI_SENZOR?op=true&quot;&lt;op name=&quot;Turn off&quot;
       href=&quot;/in-cse/in-name/TEMPERATURNI_SENZOR?op=false&quot;in=&
       quot;obix:Nil&quot; out=&quot;obix:Nil&quot; is=&quot;execute&quot;/&gt;&lt;/
       obj&gt;" +
13   "</con>" +
14 "</om2m:cin>";
15
16 HttpEntity entity = new ByteArrayEntity(
17   content.getBytes("UTF-8"));
18 request.setEntity(entity);
19 HttpResponse response = client.execute(request);
```

Izvorna koda 4.5: Ustvarjanje instance vsebnika DESCRIPTOR.

Poglavje 5

Predstavitev predlagane rešitve

5.1 Strojni del referenčne arhitekture

V tem poglavju so predstavljene strojne komponente referenčne arhitekture, ki smo jih poskusili vključiti v našo platformo. Strojni del arhitekture sestavlja ena enota IN-CSE, ena ali več enot enot MN-CSE ter naprave (mikrokontrolerji, mali računalniki), na katere so priključeni različni senzorji ali aktuatorji. Definicije in vloge enot smo predstavili že v poglavju 4.

5.1.1 Enota IN-CSE

Enota IN-CSE je lahko vsaka naprava, ki ima dovolj sistemskih virov za hranjenje ali prepošiljanje podatkov, ki jih prejme od enot MN-CSE (ali do njih dostopa). V našem primeru smo za enoto IN-CSE uporabili kar osebni prenosni računalnik. Glavni namen te enote je prejemanje in izvrševanje različnih analitičnih operacij nad podatki, pridobljenih od enot MN-CSE.

5.1.2 Enota MN-CSE

Enote MN-CSE služijo kot vozlišča, prek katerih lahko upravljamo ali pridobivamo podatke o meritvah naprav, ki komunicirajo po različnih protokolih. Prek vmesnika enote IN-CSE lahko pridemo do vmesnika vsake registrirane enote MN-CSE (viri so predstavljeni v obliki drevesne strukture), kjer si lahko ogledamo, katere

naprave so registrirane na enoti in prek katerih operacij jih lahko upravljamo ali pregledujemo vrednosti njihovih opravljenih meritev. Ker ta enota ni namenjena izvajanju različnih analitičnih operacij, je običajno to naprava z manj sistemskimi viri. V našem primeru smo uporabili računalnik Raspberry Pi.

Raspberry Pi je računalnik, velik kot kreditna kartica, ki ga je razvila angleška fundacija Raspberry Pi Fundation, da bi spodbudila zanimanje za računalniško znanost v šolah. Izdanih je bilo več modelov, nazadnje model Raspberry pi 3, ki vključuje podporo za tehnologiji Wi-Fi in Bluetooth (pri prejšnjih modelih je bilo treba kupiti namenske dodatke v obliki ključev USB, ki so omogočali komuniciranje prek določenega protokola). Na njem je mogoče poganjati več distribucij operacijskega sistema Linux, prilagojen operacijski sistem Windows 10 (IoT) ter celo operacijski sistem Android. Najpogosteje se uporablja v različnih domačih projektih (domače radijske postaje, domači podatkovni strežnik, predvajalnik multimedijskih vsebin, različne rešitve za hišno avtomatizacijo), uporablja pa se tudi v šolah za poučevanje programiranja v skriptnem jeziku Python, Scratch, C. Računalnik se napaja prek vmesnika microUSB, ima štirideset splošno namenskih vhodov in izhodov, na katere lahko povežemo različne naprave ter z njimi nato komuniciramo prek različnih protokolov (I2C, SPI, RS-232). Od podobnih kartičnih računalnikov (Beagle Board Black, Odroid C2, BananaPi) se loči predvsem v veliki in aktivni skupnosti.

5.1.3 Enota AE

Enote AE predstavljajo naprave (virtualne ali fizične), ki pošiljajo vrednosti svojih meritev enotam MN-CSE oziroma jih lahko upravljamo prek uporabniškega vmesnika enote, na katero je registrirana. Napajajo se prek baterij ali pa imajo vir stalnega napajanja. Pri tehnologijah, ki se uporabljajo za grajenje zankastih omrežij (ZigBee, Z-Wave), imajo naprave, ki so priklopljene na stalen vir napajanja, tudi vlogo posrednika sporočil v omrežju. To je pomembno zato, ker lahko sporočilo potuje do naslovljenega vozlišča po več poteh (nedelujoče vozlišče ali krajša, bolj optimalna pot) in ker lahko določeno vozlišče pošlje sporočilo vozlišču, ki ni sosedno.

Mikrokontrolerjev je danes na trgu ogromno. Najbolj priljubljen je mikrokontroler Arduino, ki z veliko abstrakcije kode, knjižnicami in namenskimi do-

datki olajša povezovanje in interakcijo s perifernimi napravami na račun hitrosti (določene operacije se namesto v enem ciklu izvedejo v petdesetih). Odločili smo se, da bomo namesto mikrokontrolerja Arduino za potrebe naloge uporabili mikrokontroler NodeMCU zaradi njegove cene (nekaj evrov), velikega nabora knjižnic za povezovanje z različnimi perifernimi napravami in vgrajene podpore za brezžični internet. Mikrokontroler temelji na čipu ESP8266, najdemo pa ga v dveh različicah (NodeMCU 0.9 in NodeMCU 1.0), ki se med seboj razlikujeta v širini ter konfiguraciji pinov. Starejša različica mikrokontrolerja ima veliko slabost, saj se je zaradi prevelike širine med njenimi pini ne da vstaviti v običajno razvojno ploščo. Programiramo ga lahko na različne načine (prek ukazov AT, v programskem jeziku C, skriptnem jeziku Lua, v poenostavljeni različici skriptnega jezika Micropython ter v poenostavljeni obliki programskega jezika C, ki se uporablja pri razvojni ploščici Arduino). Mikrokontroler smo poskusili programirati v skriptnem jeziku Lua (uporablja se dogodkovno voden način programiranja) ter v standardni obliki jezika C (proceduralni način).

Za programiranje v programskem jeziku C je na voljo uradna knjižnica, ki jo je izdalo podjetje Espressif in jo lahko uporabimo skupaj z razvojnim okoljem Arduino IDE. Za programiranje v skriptnem jeziku Lua je treba pridobiti binarne datoteke, ki vsebujejo funkcionalnosti določenih programskih modulov za mikrokontroler (modul za delo z datotečnim sistemom, modul za serijsko komunikacijo UART, modul za komunikacijo prek protokola CoAP, modul za poenostavljeno branje temperaturnih senzorjev DHTXX [29] ter mnoge druge). Binarne datoteke lahko najlažje pridobimo prek spletne storitve [16], ki nam omogoča, da si izdelamo prilagojene binarne datoteke ter na tak način prihranimo prostor. Ko imamo binarne datoteke, jih lahko naložimo z orodjem NodeMCU Flasher [27]. Ko so binarne datoteke naložene, lahko začnemo programirati v ustreznem razvojnemu okolju (najbolj razširjeno se imenuje ESPlorer [26]). Pogosta praksa pri programiranju mikrokontrolerjev NodeMCU v skriptnem jeziku Lua je, da program funkcionalno razdelimo na več datotek. Prva datoteka bo vsebovala podatke za vzpostavitev povezave z omrežjem, druga lahko vsebuje programsko kodo za povezavo z omrežjem, tretja pa vsebuje glavni del programa, ki se neskončno ponavlja (branje vrednosti določenega vhoda, na katerega je povezan senzor). Ko se mikrokontroler zažene, najprej preveri, ali se v njegovem datotečnem sistemu na-

haja datoteka init.lua. Če datoteko najde, izvede ukaze, ki se nahajajo v njej. Če datoteka vsebuje napake, ki lahko povzročijo neskončne zanke (ponovne zagone mikrokontrolerja), lahko uporabnik izbriše in ponovno naloži popravljeno datoteko. Naslednja različica mikrokontrolerja, imenovana ESP32, izide septembra 2016 in bo poleg protokola Wi-Fi podpirala tudi protokol Bluetooth in bo imela večjo velikost spomina flash. Za tiste, ki bi se radi izognili konfiguraciji, ki je potrebna za programiranje teh mikrokrmilnikov, je na voljo slika aplikacije Docker [18].

5.1.4 Senzorji in aktuatorji

Med testiranjem mikrokonrolerjev in delovanjem različnih protokolov smo na mikrokontrolerje NodeMCU in Arduino povezali številne senzorje in aktuatorje [11]. Skupaj je bilo sestavljenih deset enot.

Prvo enoto so sestavljali mikrokontroler NodeMCU, senzor svetlobe in senzor vlage (komunikacija prek protokola MQTT s pomočjo knjižnice PubSubClient [30]). Za zaznavanje intenzitete svetlobe smo uporabili fotocelico. Za zaznavanje vlage smo uporabili senzor iz družine DHT, ki poleg vlage meri tudi temperaturo.

Drugo enoto sta sestavljala mikrokontroler NodeMCU in senzor gibanja (komunikacija prek protokola MQTT). Za zaznavanje gibanja smo uporabili pasivni infrardeči senzor z oznako HC-SR501. Na senzorju sta dva upora, s katerima lahko nastavljamo doseg zaznavanja (5-10 metrov) in čas, po katerem se senzor povrne v začetno stanje (3 sekunde-300 sekund). Na sprednji strani senzorja so številne fresnerjeve leče, ki usmerjajo snop svetlobe na zaznavne elemente znotraj senzorja. Zaradi takšne porazdelitve leč senzor najbolje zaznava premikanje, ko je montiran na stropu.

Tretjo enoto sta sestavljala mikrokontroler NodeMCU in senzor razdalje (komunikacija prek protokola HTTP). Za merjenje razdalje smo uporabili senzor HC-SR04, ki omogoča zaznavanje razdalje na intervalu od dveh centimetrov do štirih metrov. Senzor meri razdaljo po principu odboja zvoka (pošlje se osem kratkih ultrazvočnih signalov na frekvenci 40 kHz, senzor pa drži pin ECHO v stanju HIGH, dokler se signali ne vrnejo).

Četrto in peto enoto sta sestavljala mikrokontrolerja Arduino in čipa za radijsko komunikacijo nRF24L01 (komunikacija prek protokola RF 2.4GHz). Na eni

enoti smo oddajali meritve svetlobnega senzorja (senzor), drugo smo uporabljali za prižiganje in ugašanje diode LED, ki je bila nameščena na mikrokontroler Arduino (aktuator). Domet čipov pri največji oddajni moči je bil 30 metrov.

Šesto enoto so sestavljali mikrokontroler NodeMCU, senzor vlažnosti zemlje in brenčač, aktivni zvočnik (komunikacija prek protokola HTTP). Večina senzorjev, ki smo jih uporabili v naših enotah in merijo določene analogne veličine, deluje po principu spreminjanja notranje upornosti. Na tak način deluje tudi senzor za zaznavanje vlažnosti zemlje.

Sedmo enoto sta sestavljala mikrokontroler NodeMCU in rele, ki je prižigal svetilko priključeno na napetost 230 V (komunikacija prek protokola MQTT). Rele je elektromehansko stikalo, ki ob prisotnosti električnega toka (magnetnea polja) sklene ali razdre stikalo v njegovi notranjosti.

Osmo enoto sta sestavljala mikrokontroler NodeMCU in rele, ki je prižigal svetilko, priključeno na napetost 230 V (komunikacija prek protokola COAP).

Deveto enoto sta sestavljala mikrokontroler NodeMCU in magnetni senzor (komunikacija prek protokola MQTT).

Deseto enoto so sestavljali mikrokontroler Arduino, senzor za zaznavo plina CO2 in senzor za zaznavo plamena (komunikacija prek protokola Bluetooth). Modul za zaznavo plina potrebuje minuto ali dve, da se segreje in začne delovati optimalno. Ko zazna prisotnost plina CO2, vrednosti sunkovito poskočijo ter se nato postopoma vrnejo v začetno stanje.

5.1.5 Drugi dodatki

Poleg različnih senzorjev, aktuatorjev in mikrokontrolerjev smo pri testiranju vključevanja naprav uporabljali tudi različne električne komponente (tranzistorje, kondenzatorje, upore), razvojne plošče, razširitvene ključe USB. Zadnji zagotavljajo komuniciranje prek določene tehnologije (ključ za brezžično internetno povezavo, ključ za komuniciranje po protokolu Bluetooth, upravljalci protokola Z-Wave).

Med enotami, ki smo jih postavili v laboratoriju, sta bili tudi dve, ki sta za komuniciranje uporabljali čipe NRF24L01 in knjižnico TMRh20 [17]. Ti čipi so postali popularni zaradi njihove cene (s Kitajske ga lahko naročimo za manj kot en evro), dobrega dometa (okoli 30 metrov, pri polni moči oddajanja) ter majhne porabe energije v primerjavi s čipi Wi-Fi. Na trenutke lahko čip porabi več toka,

kot mu ga lahko dostavi napajalni pin. V teh primerih se pojavijo napake pri prenosu, ki jih lahko odpravimo tako, da med napajalni pin in zemljo prispajkamo ustrezen kondenzator ali pa kupimo namenski dodatek (angl. *base module*), ki že ima vgrajene kondenzatorje in regulator napetosti. Čip je poleg mikrokontrolerja Arduino glavni element projekta MySensors [36], katerega glavni namen je izdelati lastne enote, na katere povežemo želene senzorje ali aktuatorje. Na strani projekta najdemo vnaprej pripravljene skripte in navodila za izdelavo glavnega vmesnika (dodatna enota, ki prejema sporočila od vseh drugih enot). Takšen pristop (uporaba osrednje enote) imajo tudi platforme, ki smo jih opisali v poglavju 2. Za implementacijo vtičnika IPU, ki bi za komuniciranje uporabljal ta čip, bi lahko uporabili knjižnico Serial nRF24L01 [37].

5.2 Programski del referenčne arhitekture

Programski del arhitekture obsega specifične vtičnike IPU in njihove odjemalce za posamezne protokole ter skripte za mikrokontrolerje NodeMCU v programskem jeziku C in Lua. Pri izdelavi vtičnikov IPU so nam bile v veliko pomoč že izdelane knjižnice za komuniciranje po določenem protokolu. Pred izdelavo vtičnikov za platformo OM2M smo v Laboratoriju za podatkovne tehnologije postavili deset enot, s katerimi je bilo mogoče komunicirati v petih različnih protokolih (HTTP, CoAP, MQTT, Bluetooth ter s čipi nRF24L01, radijske komunikacije). S postavitvijo teh enot smo lahko ugotovili, katere protokole bi bilo mogoče vključiti v našo arhitekturo in katerih ne. Ker je platforma OM2M implementirana v programskem jeziku Java, so tudi vsi vzorčni primeri kode predstavljeni v tem jeziku.

5.2.1 Vtičnik za protokol HTTP

Vtičnik IPU za protokol HTTP je od aprila 2016 privzeto na voljo v obeh različicah platforme in je prvi vtičnik, ki smo ga preučili, zato da smo ugotovili, kakšno strukturo imajo splošni vtičniki. Pri pisanju so avtorji upoštevali specifikacije standarda OneM2M za protokol HTTP [7], ki glave zahtevkov dopolnjuje z naslednjimi polji, ki so navedena v tabeli 5.1.

Polno ime	**Oznaka**
REQUEST IDENTIFIER	X-M2M-RI
ACCEPT	Accept
CONTENT TYPE	Content-Type
CONTENT LOCATION	Content-Location
ETAG	Etag
ORIGINATOR	X-M2M-Origin
NAME	X-M2M-NM
GROUP REQUEST IDENTIFIER	X-M2M-GID
RESPONSE TYPE	X-M2M-RTU
HOST	Host
ORIGINATING TIMESTAMP	X-M2M-OT
RESULT EXPIRATION TIMESTAMP	X-M2M-RET
REQUEST EXPIRATION TIMESTAMP	X-M2M-RET
OPERATION EXECUTION TIME	X-M2M-OET
EVENT CATEGORY	X-M2M-EC
RESPONSE STATUS CODE	X-M2M-RSC
CONTENT LOCATION	Content-Location

Tabela 5.1: Dodatni atributi, ki se lahko pojavijo v glavi zahtevka HTTP.

Vtičnik je sestavljen iz naslednjih razredov:

- **Activator.java** - Skrbi za pravilno aktiviranje in deaktiviranje vtičnika.
- **RestHttpClient.java** - Vsebuje implementacijo odjemalca za protokol HTTP. Ko uporabnik v spletnem vmesniku platforme OM2M klikne na določen gumb, ta razred poskrbi za pošiljanje zahtevka.
- **RestHttpServlet.java** - Implementira strežnik, ki sprejema zahtevke HTTP in jih pretvarja v zahtevke platform OM2M.
- **HttpHeaders.java** - Definira konstante, dodatne atribute v glavi zahtevka HTTP v skladu s specifikacijami standarda OneM2M za izdelavo splošnega

vtičnika.

- **HttpParameters.java** - Definira konstante, dodatne parametre spletnega naslova po specifikacijah standarda OneM2M, ki se lahko pojavijo v spletnem naslovu.

Za testiranje vtičnika lahko uporabimo poljubnega odjemalca za protokol HTTP. Vtičnik smo najprej testirali s spletnim odjemalcem Advanced rest client [54], ki je na voljo kot vtičnik za spletni brskalnik Chrome. Pozneje smo za potrebe spoznavanja delovanja vtičnika in definiranja sporočil izdelali programskega odjemalca z uporabo knjižnice Apache HttpClient [55]. Slika 5.1 prikazuje način izdelave začetnega vsebnika z izbranim imenom naprave na platformi.

```
String AEName = "TEMPERATURNI_SENZOR";
String url = "http://127.0.0.1:8080/~/in-cse";

HttpClient client = HttpClientBuilder.create().build();
HttpPost request = new HttpPost(url);

// Dodamo ustrezne atribute v glavo zahtevka
request.addHeader("X-M2M-Origin", "admin:admin");
request.addHeader("Content-Type", "application/xml;ty=2");

String content =
"<om2m:ae xmlns:om2m=
\"http://www.onem2m.org/xml/protocols\" rn=\"" + AEname+ "\">" +
    "<api>app-sensor</api>" +
    "<lbl>Type/sensor Category/temperature Location/home</lbl>" + "<rr>true</rr
      >" +
"</om2m:ae>";

HttpEntity entity = null;

try {
  entity = new ByteArrayEntity(content.getBytes("UTF-8"));
} catch (UnsupportedEncodingException e1) {
    System.out.println("Failed to get, convert content.");
```

```
24 }
25 request.setEntity(entity);
26 response = client.execute(request);
```

Izvorna koda 5.1: Registracija naprave s sporočilom protokola HTTP.

5.2.2 Vtičnik za protokol CoAP

Ker novejša različica platforme OM2M ob začetku pisanja tega dela še ni imela podpore za protokol CoAP, smo se odločili za njegovo implementacijo. Vtičnik ima podobno strukturo kot vtičnik za protokol HTTP, saj sta si protokola v veliko zadevah enaka. Implementira osnovne operacije platforme, s katerimi lahko: prijavimo napravo na platformo, izdelamo potrebne vsebnike za napravo, iščemo naprave, se naročimo na prejemanje sporočil določene druge naprave. Pri implementaciji smo uporabili knjižnico Californium [25], s katero je bilo mogoče implementirati strežniški del in tudi odjemalca. Pri pisanju smo upoštevali specifikacije standarda OneM2M za protokol CoAP [6], ki sporočila dopolnjuje z neobveznimi polji, ki so navedena v tabeli 5.2.

Polno ime neobveznega polja	**Oznaka**	**Vrednost**
FROM	oneM2M-FR	256
REQUEST IDENTIFIER	oneM2M-RQI	257
ORIGINATING TIMESTAMP	oneM2M-OT	259
REQUEST EXPIRATION TIMESTAMP	oneM2M-RQET	260
RESULT EXPIRATION TIMESTAMP	oneM2M-RSET	261
OPERATION EXECUTION TIME	oneM2M-OET	262
NOTIFICATIONURI OF RESPONSE TYPE	oneM2M-RTURI	263
EVENT CATEGORY	oneM2M-EC	264
RESPONSE STATUS CODE	oneM2M-RSC	265
GROUP REQUEST IDENTIFIER	oneM2M-GID	266
RESOURCE TYPE	oneM2M-TY	267

Tabela 5.2: Neobvezna polja vtičnika za protokol CoAP.

Vtičnik je sestavljen iz naslednjih razredov:

- **Activator.java** - Skrbi za pravilno aktiviranje in deaktiviranje vtičnika.
- **CoapMessageDeliverer.java** - Razred, v katerem se nahaja metoda, ki preoblikuje prejeta sporočila protokola CoAP v sporočilo platforme OM2M.
- **MyCoapClient.java** - Vsebuje implementacijo odjemalca za protokol CoAP. Ko uporabnik v spletnem vmesniku platforme OM2M klikne na določen gumb, ta razred poskrbi za pošiljanje zahtevka.
- **MyCoapServer.java** - Razred, v katerem je inicializiran objekt strežnika CoAP. Ob zagonu vtičnika se kliče metoda start(), ki je definirana v tem razredu.
- **CoapNewOptions.java** - Definira konstante, dodatne atribute v glavi sporočila CoAP v skladu s specifikacijami standarda OneM2M za izdelavo splošnega vtičnika.
- **CoapParameters.java** - Definira konstante, dodatne parametre spletnega naslova po specifikacijah standarda OneM2M, ki se lahko pojavijo v spletnem naslovu.

Poleg vtičnika je bilo treba implementirati tudi preprostega odjemalca (v obliki konzolnega programa), ki je skladen s specifikacijami standarda OneM2M za protokol CoAP. Trenutne implementacije odjemalcev za protokol ne omogočajo definiranja večjega števila neobveznih polj v sporočilu CoAP. Orodje Copper sicer ima polje *custom options*, ampak se je izkazalo, da je to polje kljub imenu namenjeno samo enemu neobveznemu polju. O napaki smo obestili avtorja. Spodnja koda prikazuje, kako lahko s pomočjo knjižnice Californium ustvarimo sporočilo za ustvarjanje naprave na platformi OM2M.

```
1 String AEName = "TEMPERATURNI_SENZOR";
2 String createApplicationBody =
3 "<om2m:ae xmlns:om2m=
```

```
\"http://www.onem2m.org/xml/protocols\" rn=\"" + AEname+ "\">" +
  "<api>app-sensor</api>" +
     "<lbl>Type/sensor Category/temperature </lbl>" +
       "<rr>false</rr>" +
"</om2m:ae>";

Request createApplication = new Request(Code.POST);
createApplication.setToken(new byte[]{}); // brez polja "token"

createApplication.getOptions() // definiramo neobvezna polja
   .setContentFormat(MediaTypeRegistry.APPLICATION_XML)
   .setAccept(MediaTypeRegistry.APPLICATION_XML)
   .addOption(new Option(256, "admin:admin")) // neobvezno polje "FROM"
   .addOption(new Option(267, 2)); // neobvezno polje "RESOURCE TYPE"
createApplication.setPayload(createApplicationBody);

String coapServerAddress =  "coap://127.0.0.1:5683/~/in-cse";
CoapClient c = new CoapClient(new URI(coapServerAddress));
c.advanced(createApplication);
```

Izvorna koda 5.2: Registracija naprave s sporočilom protokola CoAP.

V različici platforme, ki jo lahko prenesemo iz njenega uradnega repozitorija, je vtičnik za protokol CoAP vključen samo v enoto IN-CSE. Če želimo vključiti ta vtičnik tudi v vmesno enoto MN-CSE, to storimo tako, kot je predstavljeno v vodiču na uradni strani [46].

5.2.3 Vtičnik za protokol MQTT

Vtičnik za protokol MQTT se od prejšnjih dveh vtičnikov razlikuje v tem, da za svoje delovanje potrebuje posrednika, katerega naloge so skrb za avtentikacijo in avtorizacij odjemalcev ter posredovanje sporočil.

Pred implementiranjem vtičnika smo namestili posrednika na računalnik Raspberry pi 3, ki se je nahajal v lokalnem omrežju, in na osebni prenosni računalnik. Odločili smo se za namestitev posrednika Mosquitto, ker podpira vse zahteve, ki so navedene v specifikaciji standarda OneM2M za protokol MQTT. Posrednika lahko konfiguriramo s pomočjo konfiguracijske datoteke mosquitto.conf, ki se nahaja v

namestitveni mapi aplikacije. Omenjeni posrednik je na voljo za različne operacijske sisteme. Na sistemih Linux ga lahko namestimo z ukazom: apt-get install mosquitto, vendar nam ta ukaz ne namesti najnovejše različice aplikacije. Pravilen postopek ročne namestitve je prikazan v izvorni kodi 5.3.

```
1  sudo apt-get update
2  sudo apt-get install libssl-dev
3  sudo apt-get install cmake
4  sudo apt-get install libc-ares-dev
5  sudo apt-get install uuid-dev
6  sudo apt-get install daemon
7  wget http://mosquitto.org/files/source/mosquitto-1.4.5.tar.gz
8  tar xzf mosquitto-1.4.5.tar.gz
9  cd mosquitto-1.4.5
10 cmake .
11 sudo make install
```

Izvorna koda 5.3: Postopek ročne namestitve posrednika Mosquitto.

Na sistemih z operacijskim sistemom Windows lahko zaženemo namestitveno datoteko [57], s katero namestimo posrednika. Za njegovo delovanje so potrebne tri datoteke DLL, ki pripadajo knjižnicama OpenSSL [58] in Pthreads [59]. Iz knjižnice OpenSSL je treba skopirati datoteki DLL z imenoma *libeay32.dll* in *ssleay32.dll*. V zadnji različici te knjižnice ti dve datoteki nista več prisotni, zato ju je najlažje poiskati na spletu. Poleg teh dveh datotek je treba prenesti še datoteko "pthreadVC2.dll", ki pripada knjižnici Pthreads. Vse tri datoteke skopiramo v namestitveno mapo posrednika Mosquitto (privzeto `C:\ProgramFiles(x86)\mosquitto`), tako da so na isti ravni kot izvršilna datoteka *mosquitto.exe*. Zatem je treba ponovno namestiti posrednika mosquitto, zato da se ustrezno konfigurira pripadajoča storitev, ki teče v ozadju. Ali je posrednik pravilno nameščen, lahko preverimo v konzoli za upravljanje vseh storitev, ki tečejo v ozadju na računalniku z operacijskim sistemom Windows - *services.msc*. Če se v konzoli nahaja vnos "Mosquitto broker", je posrednik pravilno nameščen. Dvokliknemo na ta vnos ter s klikom na gumb *start* zaženemo posrednika, ki bo deloval v ozadju. Naslov in delovanje posrednika lahko dodatno preverimo v ukazni vrstici z ukazom, ki je prikazan spo-

daj. Po izvedbi ukaza se nam izpišejo vsi vnosi, ki uporabljajo vrata 1883 (protokol Mqtt).

```
1 netstat -an | find "1883"
```

Poleg posrednika smo uporabili knjižnico Eclipse Paho, s katero je bilo mogoče implementirati odjemalca. Za tega je treba implementirati tri metode povratnih asinhronih klicev, ki določajo njegovo delovanje. Prva metoda, *connetctionLost*, definira obnašanje odjemalca po izgubi povezave s posrednikom. Druga metoda, *messageArrived*, definira obnašanje posrednika ob prejemu sporočila. Pri implementaciji te metode je treba biti pazljiv, ker se odjemalec v primeru proženja izjeme zaustavi. Tretja metoda, *deliveryComplete*, definira obnašanje odjemalca po prejemu potrdila, da je bilo njegovo sporočilo prejeto.

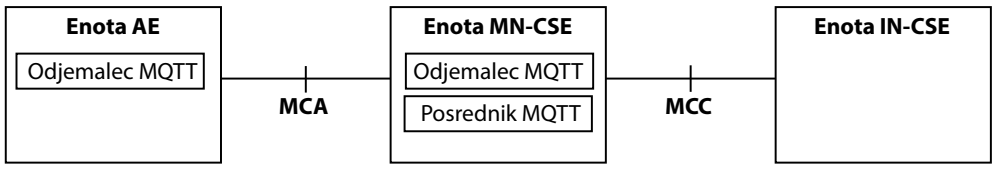
Odjemalec je sestavljen iz naslednjih razredov:

- **MyMQTTClient.java** - Skrbi za zagon odjemalca. V tem razredu se nahajajo metode za zagon odjemalca, metodi za povezovanje in prekinitev povezave s posrednikom, metoda za objavo sporočila in naslov posrednika.

- **Messages.java** - Razred vsebuje vsebino teles sporočil, ki jih pošlje odjemalec.

- **CallbackMethods.java** - Razred vsebuje definicije treh metod povratnih asinhronih klicev, ki so že bili opisani.

V specifikacijah standarda OneM2M je za protokol MQTT predstavljenih več konfiguracij omrežij, ki se med seboj razlikujejo glede na lokacijo strežnika MQTT, ter protokoli za komuniciranje med napravami. Slika 5.1 prikazuje konfiguracijo, ki smo jo uporabili v naši arhitekturi.



Slika 5.1: Konfiguracija omrežja pri protokolu MQTT.

Pri tej konfiguraciji se vsaka naprava povezuje na komponento MN-CSE prek protokola MQTT, komponenta MN-CSE pa se do komponente IN-CSE povezuje prek nekega drugega protokola (v našem primeru je to protokol HTTP). Organizacija omrežja, opis komunikacije med odjemalci in predstavitev osnovnih konceptov so opisani v specifikacijah standarda OneM2M za protokol MQTT [8].

Vtičnik je sestavljen iz naslednjih razredov:

- **Activator.java** - Skrbi za pravilno aktiviranje in deaktiviranje vtičnika.
- **MqttMessageDeliverer.java** - Razred vsebuje definicije metod asinhronih povratnih klicev, ki definirajo obnašanje strežnika ob izgubi povezave, končanem prenosu sporočila, prejemu sporočila.
- **MyMqttClient.java** - Razred se lahko uporabi za implementacijo odjemalca (upravljanje naprav prek gumbov v spletnem vmesniku platforme), ki ga v okviru naloge nismo implementirali.
- **MyVirtualMqttServer.java** - Razred vsebuje metode za delo s strežnikom (zagon, zaustavitev, povezava do posrednika).

Specifikacije tega protokola določajo, da je treba v vtičnik vključiti klienta, ki bo imel vlogo strežnika. Ta klient se mora po zagonu vtičnika povezati s posrednikom ter upoštevati naslednji format naslovov tem za objavljanje sporočil:

- Tema **/oneM2M/req/<pošiljatelj>/<prejemnik>** se uporablja za pošiljanje zahtevkov za pridobivanje nekega vira. Naslov teme vsebuje enolična identifikatorja pošiljatelja sporočila in prejemnika sporočila.

- Tema **/oneM2M/req/+/<prejemnik>** omogoča prejemanje sporočil od poljubnega pošiljatelja. To omogoča maskirni znak "+" v naslovu teme.
- Tema **/oneM2M/resp/<pošiljatelj>/<prejemnik>** je namenjena pošiljanju odzivov na prejeta sporočila.

Preden je odjemalec registriran na platformi, se mora izvesti postopek začetne registracije, med katerim odjemalcu platforma dodeli enolični identifikator. V ta namen so definirani še dve temi z naslednjim formatom, na kateri se prijavijo novi odjemalci, ter strežnik:

- Tema **/oneM2M/req_req/<pošiljatelj>/<prejemnik>** se uporablja za pošiljanje zahtevkov začetne registracije.
- Tema **/oneM2M/req_resp/<pošiljatelj>/<prejemnik>** se uporablja za prejem zahtevkov začetne registracije.

```
1  MqttClient client;
2  String BROKER_URI = "tcp://192.168.0.202:1883";
3  boolean CLEAR_SESSION_FLAG = false;
4  String CLIENT_ID = "AE-1234";
5  String TOPIC = "/oneM2M/reg_req/" + CLIENT_ID + "/in-cse";
6
7  public static String createAMySensorApplication =
8   "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
9    +"<m2m:rqp xmlns:m2m=\"http://www.onem2m.org/xml/protocols\">"
10      + "<op>1</op>"
11      + "<to>~/in-cse</to>"
12      + "<fr>admin:admin</fr>"
13      + "<ty>2</ty>"
14      + "<nm>TEMPERATURNI_SENZOR</nm>"
15      + "<pc>"
16       + "<om2m:ae xmlns:om2m=\"http://www.onem2m.org/xml/protocols\">"
17        + "<api>app-sensor</api>"
18        + "<lbl>Type/sensor Category/temperature</lbl>"
19        + "<rr>false</rr>"
20       + "</om2m:ae>"
21      + "</pc>"
```

```
22     + "</m2m:rqp>";
23
24 client = new MqttClient(brokerAddress, clientId);
25
26 // Ustrezno nastavimo zastavice
27 MqttConnectOptions options = new MqttConnectOptions();
28
29 // Zastavico "Clean session" nastavimo na vrednost "false"
30 options.setCleanSession(CLEAR_SESSION_FLAG);
31 // Onemogocimo Keep alive mehanizem ali pa
32 // ga nastavimo na doloceno vrednost (v sekundah)
33 options.setKeepAliveInterval(0);
34 //options.setKeepAliveInterval(10);
35
36 // omogocimo prejem sporocil
37 client.setCallback(new CallbackMethods());
38
39 // Posljemo sporocilo tipa CONNECT (Id = 0)
40 client.connect(options);
41
42 client.publish(TOPIC, message.getBytes(), 1, false);
```

Izvorna koda 5.4: Registracija naprave s sporočilom protokola HTTP.

Pri vtičniku za ta protokol se pri prenosu sporočil prenaša celotno serializirano sporočilo, ki je lahko v formatu XML ali formatu JSON. Med izdelavo je bilo ugotovljeno, da trenutna implementacija platforme OM2M ne zagotavlja pravilne (de)serializacije sporočila v sporočilo, ki bi ustrezalo standardu OneM2M, saj izpusti del z oznako *Content*. V naši implementaciji smo to težavo rešili tako, da pred deserializacijo prejetega sporočila preverimo, ali sporočilo vsebuje del *Content*. Če ga, si ga zapomnimo, preostali del sporočila pa deserializiramo v sporočilo platforme OM2M. Zapomnjeni del nato vstavimo v objekt deserializiranega sporočila, tako da dobimo celotno sporočilo. O tej težavi smo pisali tudi avtorjem platforme na uradnem forumu.

Za avtentikacijo in avtorizacijo odjemalcev, ki želijo komunicirati z napravami, je zadolžen strežnik MQTT, ki ga je treba ustrezno konfigurirati. V nekaterih primerih odjemalci ob prvotni registraciji prejmejo začasen enolični identifikator,

pozneje pa jim posrednik dodeli ustrezen nov identifikator.

5.2.4 Vtičnik za protokol Z-Wave

Pred začetkom implementacije vtičnika za protokol Z-Wave smo raziskali, katere knjižnice so na voljo za ta protokol v programskem jeziku Java. Odkrili in preizkusili smo naslednje:

- WZWave [14],
- ZWave Library for Java [20],
- ZWave4j [21].

Pri pregledu knjižnic smo ugotovili, da prvi dve ne podpirata programskega proženja postopka vključevanja naprav v omrežje Z-Wave, saj predvidevata, da bo uporabnik uporabljal upravljalca z gumbom za ročno proženje postopka vključitve. Na trgu lahko najdemo dve taki napravi: AEON Labs Series2 in AEON Labs GEN5. Razlika med ključema je v tem, da model GEN5 podpira tudi naprave Z-Wave Plus (izboljšana življenjska doba baterije, izboljšan domet, večja pasovna širina, podpora metodi za oddaljeno nadgrajevanje strojne opreme, podpora kanalom), poleg tega vključuje orodje za diagnosticiranje stanja omrežja na podlagi barvnega kodiranja utripajoče diode LED. Oba ključa uporabljata uradno knjižnico SerialAPI, ki je v lasti podjetja ZenSys in je na voljo samo partnerjem podjetja. To tudi pomeni, da lahko ključe uporabljamo s preostalimi orodji, ki to knjižnico potrebujejo za delovanje.

Prvi dve knjižnici implementirata samo nekaj razredov za delo z napravami, poleg tega pa smo pri preizkušanju ugotovili, da se pri pošiljanju ukazov pojavljajo moteče zapoznitve. Pred testiranjem knjižnice WZWave smo z avtorjem knjižnice najprej odpravljali napako odkrivanja naprav Z-Wave plus. Aplikacija ob zagonu ni zaznala vseh vozlišč, ki so bila vključena v omrežje. Naključno se je tudi dogajalo, da ukazi niso bili poslani, kar smo preverili s posebnim ključem USB za spremljanje prometa. Modul ne prejme ukaza v enem od petnajstih poskusov, zaradi česar je knjižnica nezanesljiva in tudi neprimerna za uporabo z varnostnega stališča (v primeru zapiranja oken, žaluzij, gretja).

Knjižnica Zwave library for java je napisana na funkcijski način in za delovanje potrebuje najnovejšo različico Jave (1.8). Implementira nekaj osnovnih razredov za delo z napravami (BASIC, BINARY_SWITCH, METER). Tudi pri tej knjižnici smo opazili nekaj težav pri pošiljanju in odzivanju naprav na poslane ukaze.

Na koncu smo se odločili, da bomo pri implementaciji vzorčnega vtičnika uporabili knjižnico Zwave4j, saj pri testiranju nismo zasledili nobenih zapoznitev, knjižnica ima implementiranih največ funkcionalnosti protokola Z-Wave (asociacije, scenariji, podpira razred COMMAND_CLASS_SECURITY), poleg tega pa je na voljo dokumentacija [38], ki ju preostali dve knjižnici nimata. Dodatna prednost te knjižnice je možnost programskega proženja vključevanja vozlišč v omrežje. Knjižnica ovija knjižnico OpenZWave, ki je implementirana v programskem jeziku C++, kar tudi pojasni odsotnost zapoznitev, ki smo jih opazili pri drugih dveh knjižnicah. Pomembnejše metode, ki smo jih zasledili v dokumentaciji:

- **SetNodeOn** - Ta metoda se uporablja za opravljanje osnovne funkcionalnosti naprave (metode BASIC_SET(0xFF) razreda BASIC). V primeru žarnice bi ta metoda prižgala luč.

- **SetNodeOff** - Podobno kot prejšnja metoda ta povzoči, da se žarnica ugasne (pošlje se BASIC_SET(0x00)).

- **SetNodeLevel** - Ta metoda se uporablja za upravljanje naprav, ki lahko za upravljanje sprejemajo vrednosti na določenem intervalu (običajno je to interval [0x00 - 0xFF]). V primeru žarnice, ki lahko sprejema več vrednosti, bi ukaz SetNodeLevel(50) povzročil, da bi se žarnica prižgala na 50 % svetlosti.

- **SetConfigurationParameter** - Konfiguracijski parametri omogočajo spreminjanje delovanja naprav Z-Wave (poročanje v izbrani enoti, nastavljanje različnih časovnih intervalov, nastavljanje možnosti vhodov in izhodov naprave, nastavljanje različnih pragov za proženje obvestil). Ta metoda omogoča nastavljanje vrednosti izbranega parametra.

- **RequestConfigParam** - S to metodo pridobimo vrednost konfiguracijskega parametra.

- **GetNodeClassInformation** - S to metodo lahko upravljalec vpraša določeno napravo ali implementira funkcionalnosti določenega razreda.
- **RefreshValue** - Če želimo napravo vprašati samo enkrat za njeno stanje ali izbrano vrednost, uporabimo to metodo.
- **EnablePoll** - S to metodo omogočimo periodično pridobivanje vrednosti (angl. *polling*), ki jo določa podani objekt ValueId.

Ideja vtičnika je, da gre uporabnik najprej po hiši s ključem ter v omrežje vključi vse naprave, ki komunicirajo po protokolu Z-Wave. Ko ima vse naprave zbrane, aktivira vtičnik platforme OM2M ter pogleda, ali so na ključu shranjene kakšne naprave. Če so, za vsako ustvari ustrezne operacije v grafičnem vmesniku platforme. Med intervjujem naprave upravljalec odkrije tudi vrednosti, za katere lahko vpraša posamezno napravo. Vrednosti so predstavljene z objekti ValueId, ki so sestavljeni iz številnih atributov. Ob zagonu vtičnika se sproži tako imenovani postopek intervjuja naprav, med katerim upravljalec zgradi datoteko XML, ki vsebuje podatke o omrežju in vključenih vozliščih. Ime datoteke se začne s predpono 'zwcfg_', ki ji sledi vrednost `HOME_ID`. Ta format se uporablja za zagotavljanje unikatnosti konfiguracijske datoteke, saj je lahko na napravo priključenih več upravljalcev. Datoteka se ustvari (ali posodobi, če je že ustvarjena) ob zagonu vtičnika in ob njegovem izklopu. Ko se intervju konča, se sproži metoda povratnega asinhronega klica, ki platformi sporoči, da so bili pridobljeni vsi podatki o napravah, zato se lahko na platformi zanje ustvari ustrezne vsebnike. V vtičniku smo implementirali tudi programsko vključevanje in odstranjevanje naprav iz omrežja.

Vtičnik je sestavljen iz naslednjih razredov:

- **Activator.java** - Glavni razred, ki skrbi za aktivacijo vtičnika in začetno vzpostavitev omrežja. V tem razredu sta definirani dve pomembni spremenljivki, ki sta potrebni za delovanje knjižnice. Prva, imenovana pathToOpenZWaveConfigDir, kaže na pot do konfiguracijske mape knjižnice OpenZWave, v kateri so podatki o proizvajalcih in njihovih napravah (predstavljene v formatu XML). Če določene naprave, ki jo želimo uporabiti, ni v omenjeni mapi, je treba zanjo ustvariti konfiguracijsko datoteko. Običajno gre za predelane datoteke XML, ki jih proizvajalci posredujejo na stran Pepper1 [48],

ki vsebuje bazo naprav Z-Wave. Druga spremenljivka označuje serijski vmesnik (port COM), na katerega je priklopljen naš upravljalec. Ta korak bi se dalo avtomatizirati tako, da bi s pomočjo ustrezne knjižnice pridobili zbirko serijskih vmesnikov in njihovih imen (angl. *friendly name*). Če bi bilo ime nekega vmesnika enako nizu "UZB" (v primeru upravljalca AEON GEN5), bi izbrali ta vmesnik.

- **CommandClassLabels.java** - Knjižnica OpenZwave nikjer ne prikazuje imen podprtih razredov, ampak izpisuje samo njihove oznake, ki so predstavljene z enim bajtom v heksadecimalni obliki. Namen tega razreda je preslikati oznake razredov v človeško berljiva imena razredov.

- **ContainerCreator.java** - Razred, katerega naloga je ustvarjanje vsebnikov na platformi.

- **Controller.java** - Razred skrbi za ustrezno preusmerjanje zahtevkov, ki jih uporabnik proži prek uporabniškega vmesnika platforme, in izvajanje operacij s pomočjo glavnega (angl. *singleton*) objekta - manager.

- **ObixUtil.java** - Razred implementira funkcije, ki omogočajo predstavitev naprav v formatu OBiX. Naprave so predstavljene s številnimi vsebniki. Vsebnik `DESCRIPTOR`, tako kot pri drugih vtičnikih, vsebuje metapodatke o napravi. V ta vsebnik so bili dodani tudi gumbi, ki ustrezajo objektom ValueId in s katerimi lahko napravo vprašamo po določeni vrednosti, ki jo oglašuje. Poleg tega vsebnika je za vsak podprt razred ustvarjen vsebnik z imenom razreda. Namen teh vsebnikov je hranjenje vrednosti istoimenskih razredov.

- **Pair.java** - Pomožni razred, ki predstavlja objekt za hranjenje para (ključ in vrednost).

- **RequestSender.java** - Pomožni razred, ki olajša grajenje poizvedb platforme OM2M. Razred vsebuje različne metode, kot so na primer: metoda za ustvarjanje vsebnikov in njihovih instanc, metoda za brisanje naprave s platforme, metoda za preverjanje obstoja naprave z določenim imenom na platformi.

- **ValueObj.java** - Objekt vsebuje vse atribute razreda ValueId. Vsebuje dva dodatna atributa, s katerima dodatno opišemo predstavljeno vrednost. Prvi atribut je namenjen heksadecimalni predstavitvi razreda (platforma privzeto prikazuje samo decimalne vrednosti razredov), drugi pa je namenjen hranjenju naziva razreda, ki mu vrednost pripada.
- **ZWaveNetwork.java** - Razred, v katerem hranimo različne podatke o omrežju Z-Wave. Vsebuje seznam vozlišč, ki so vključena v omrežje, in seznam vrednosti (objekti ValueId), za katere lahko napravo vprašamo.
- **ZWaveNode.java** - Razred, namenjen predstavitvi posameznega vozlišča.

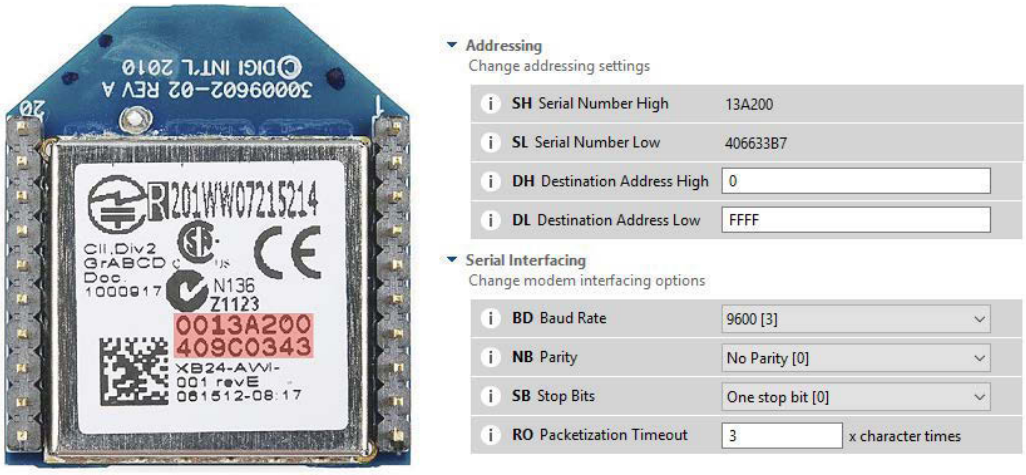
Med izdelavo vtičnika za platformo smo naleteli na problem nepravilne inicializacije knjižnice Zwave4j, zaradi katere ni bilo mogoče poganjati vtičnika. Odkrili smo, da sta problem in njegova rešitev že opisana na strani knjižnice [44].

Pri pregledu sorodnih platform smo ugotovili, da večina platform uporablja knjižnico OpenZWave [35] ali knjižnico ZWaveMe [49]. Za uporabo zadnje je potreben ustrezen upravljalec, poleg tega pa je treba za uporabo vseh funkcionalnosti kupiti licenco. Platforma OpenHAB ima lasten vtičnik za ta protokol, platforma DomotiGa pa uporablja knjižnico Zwave4j.

5.2.5 Vtičnik za protokol ZigBee

Implementirali smo primer vzorčnega vtičnika za protokol ZigBee, ki lahko prejema meritve od različnih modulov, na katere so povezani senzorji. Moduli XBee omogočajo pošiljanje stanj svojih vhodov in izhodov, ne da bi bila nanje povezana dodatna naprava. Če bi želeli vključiti dodatno programsko logiko, bi bilo treba na modul povezati določeno napravo (na primer mikrokontroler Arduino ali mikrokontroler AVR). Vtičnik smo preizkusili na dveh modulih (XBee S5 Pro), ki sta bila na voljo in ju je bilo treba pred izdelavo vtičnika ustrezno nastaviti. Za nastavljanje smo uporabili orodje X-CTU, ki predstavlja grafični vmesnik za konfiguriranje modulov z ukazi AT (angl. *ATtention*), ki so se v preteklosti uporabljali med drugim tudi za konfiguriranje telefonskih modemov in mobilnih telefonov. Nabor ukazov AT, ki jih lahko pošljemo modulu, določa njegov tip. Enemu modulu smo nastavili vlogo koordinatorja omrežja, drugemu vlogo usmerjevalnika.

Koordinatorju je bilo treba nastaviti način delovanja na API 2, saj to za delovanje potrebuje knjižnica XBee-api [28]. Ta način delovanja lahko nastavimo v orodju X-CTU tako, da v spustnem polju, označenem z "AP - API Enable", izberemo vrednost "API with escapes 2". Usmerjevalniku smo nastavili naslov prejemnika na naslov koordinatorja MAC. Če tega naslova ne bi nastavili, bi usmerjevalnik pošiljal sporočila na naslov broadcast, kar bi, če bi imeli veliko število modulov, ustvarjalo nepotreben promet. Vsak modul XBee je enolično določen z dvema skupinama številk, ki skupaj tvorita njegov naslov MAC.



Slika 5.2: Unikaten naslov modula MAC in nastavitve konfiguracije.

Slika 5.2 prikazuje, kako je ta naslov prikazan na posameznem modulu in s katerima dvema poljema ga lahko nastavimo v orodju X-CTU. Poleg enoličnega naslova so prikazane tudi nastavitve parametrov serijske komunikacije, ki morajo biti enake na vseh modulih.

Za izmenjevanje sporočil med moduli smo si zamislili lastno stukturo sporočil, ki so predstavljena v formatu JSON. V izvorni kodi 5.5 sta prikazani dve sporočili, s katerima lahko registriramo napravo in pošiljamo meritve na platformo. Ker na modul nismo priklopili senzorja, s katerega bi brali vrednosti, smo vrednosti meritev simulirali z uporabo konzole, ki se nahaja v orodju X-CTU in omogoča pošiljanje sporočil (če bi na modul povezali določen senzor, bi lahko na modulu z ukazi ATDX, ATPX, ATIR, ATWR nastavili periodično pošiljanje meritev). Do

konzole pridemo tako, da v glavnem meniju orodja kliknemo na gumb, na katerem je ikona ekrana, nato pa še na gumb, s katerim odpremo lokalno serijsko povezavo med našim računalnikom in modulom.

```
1 {"op": "createAE", "nm": "XBEE_MODUL"}
2 {"op": "dataCI", "nm": "XBEE_MODUL", "val": "23" }
```

Izvorna koda 5.5: Primer serializiranega sporočila.

Vtičnik je sestavljen iz naslednjih razredov:

- **Activator.java** - Skrbi za pravilno aktiviranje in deaktiviranje vtičnika.
- **Monitor.java** - V tem razredu je definirana metoda povratnega asinhronega klica, ki neprestano posluša za nova prispela sporočila. Ko sporočilo prispe, ta razred poskrbi za njegovo razlčlenitev, preoblikovanje v ustrezno sporočilo platforme OM2M in izvršitev. V tem razredu (vrstica 122) je treba tudi nastaviti vrata, na katera je priklopljen modul, ki ima vlogo koordinatorja omrežja in baudno hitrost, ki mora biti enaka za vse module v omrežju.
- **ObixUtil.java** - Naprave so v platformi OM2M privzeto predstavljene v formatu oBIX [39], ki predstavlja standardizirano obliko sporočil XML za opisovanje stavb. Razred omogoča dinamično grajenje opisov naprav v tem formatu.
- **RequestSender.java** - Podobno kot pri drugih vtičnikih najdemo v tem razredu metode, ki olajšajo grajenje poizvedb platforme OM2M.

5.2.6 Druge programske rešitve

V sklopu preučevanja različnih naprav, ki jih lahko vključimo v platformo, smo preučili tudi možnost vključitve pametnih telefonov z operacijskim sistemom Android. Ti so zelo primerni za vključitev zaradi mobilnosti, možnosti nalaganja lastnih aplikacij in ker vsebujejo številne senzorje, ki jih lahko izkoristimo za spremljanje in merjenje veličin v naši okolici (senzor temperature, vlage, senzor oddaljenosti, žiroskop, kompas).

Zanimalo nas je, kako bi lahko pametni telefon uporabili kot razširjeni vmesnik GSM. Funkcionalnosti običajnih vmesnikov GSM, ki jih najdemo v obliki namenskih modulov za priklop na ustrezne naprave (na primer mikrokontroler Arduino), so posredovanje sporočil SMS in proženje klica na vnaprej določeno telefonsko številko. Izdelali smo primer aplikacije, ki predstavlja razširjeni vmesnik GSM ter omogoča pošiljanje in prejemanje sporočil SMS, pošiljanje in prejemanje telefonskih klicev, pošiljanje in prejemanje podatkov prek povezave Bluetooth, pošiljanje zahtevkov HTTP, naročanje in pošiljanje sporočil protokola MQTT (s knjižnico Paho). V aplikacijo je mogoče vključiti tudi podporo za protokol CoAP s pomočjo knjižnice Californium [50]. Aplikacija se ob proženju določenega dogodka (prejem sporočila SMS, prejem klica, prejem sporočila MQTT, prejem sporočila prek protokola Bluetooth) ustrezno odzove (vrsto odziva izbere uporabnik).

Poglavje 6

Evaluacija

V uvodnih poglavjih smo predstavili sorodne platforme, protokole in na splošno predstavili zveze in asociacije, ki delujejo na področju standardizacije IoT.

Praktični del smo začeli s spoznavanjem mikrokontrolerja NodeMCU, ki ga lahko programiramo v številnih programskih in skriptnih jezikih. V Laboratoriju za podatkovne tehnologije smo postavili deset enot, s katerimi je bilo mogoče komunicirati v petih različnih protokolih (MQTT, CoAP, HTTP, Bluetooth, RF 2.4GHz). Med testiranjem delovanja mikrokontrolerjev, ki so bili programirani v skritpnem jeziku Lua, smo opazili, da so se nekajkrat samodejno resetirali zaradi pomanjkanja pomnilnika in so se počasneje odzivali na prejete ukaze. Nasprotno pri mikrokontrolerjih, ki so bili programirani v programskem jeziku C, teh problemov nismo zasledili.

Sledilo je spoznavanje platforme OM2M in specifikacij posameznih protokolov standarda OneM2M. Pred začetkom implementacije vtičnikov za platformo OM2M smo raziskali, katere protokole bi bilo mogoče vključiti v platformo. Upoštevati je bilo treba, da je vtičnik moral delovati na napravah z operacijskima sistemoma Windows in Linux. Izkazalo se je, da je večina protokolov dobro podprta v programskem jeziku Java.

Nadaljevali smo z izdelavo splošnih vtičnikov za protokola CoAP in MQTT. Za zadnjega smo implementirali samo pošiljanje vrednosti. V obeh vtičnikih so implementirane osnovne operacije, ki so v poglavju 4.5 predstavljene za protokol HTTP. Za testiranje splošnih vtičnikov za protokola CoAP in MQTT je bilo treba izdelati odjemalca, ki sta z vtičnikoma komunicirala v skladu s specifikacijami stan-

darda OneM2M. Pri izdelavi odjemalca za protokol CoAP smo uporabili knjižnico Californium, za izdelavo odjemalca za protokol MQTT pa knjižnico Paho. Za testiranje vtičnika za protokola HTTP in CoAP smo priredili odjemalca tako, da vsak od njiju pošlje tisoč zahtevkov na minuto. Ker mora vsak poslani zahtevek po specifikacijah standarda OneM2M prejeti odziv z ustrezno kodo, smo za merilo uspešnosti izbrali število prejetih odzivov, ki potrjujejo prejem in uspešno izvedbo. To je pri protokolih HTTP, CoAP in MQTT zagotovljeno v obliki šifrantov (201 pri HTTP, 2.01 za CoAP, 2001 pri protokolu MQTT). Z obema odjemalcema smo v zanki poskusili ustvariti tisoč virtualnih naprav. Na platformo je bilo v petih minutah poslanih deset tisoč zahtevkov, od katerih so vsi prejeli odziv z ustreznim šifrantom. Delovanje vtičnika za protokol HTTP smo preizkusili tudi na treh mikrokontrolerjih NodeMCU, na katere smo povezali določene aktuatorje. S pomočjo preusmeritvenih zahtevkov smo zanje na platformi ustvarili ustrezne vsebnike ter vmesnike do njihovih metod za proženje.

Pri testiranju vtičnika za protokol MQTT smo ugotovili, da je trenutna implementacija vtičnika sposobna prejemati sporočila v razmiku 200 ms. Razlog za to je, da trenutna implementacija metode za obdelavo prejetega sporočila prejetih sporočil ne shranjuje v ustrezno podatkovno strukturo (na primer vrsto), ampak poskuša nanje odgovoriti takoj, ko jih prejme. Naprave, ki uporabljajo splošne vtičnike platforme OM2M za komuniciranje po določenem protokolu, se lahko naročijo na obvestila o prejemanju sporočil druge naprave (angl. *subscriptions*). Za potrebe testiranja te funckionalnosti smo pri protokolu CoAP izdelali aplikacijo, ki na določenem naslovu neprestano posluša za takšna obvestila.

Vtičnik za protokol ZigBee smo testirali na dveh modulih XBee (serije S5). Prvi modul je imel vlogo koordinatorja, drugi pa vlogo usmerjevalnega vozlišča. Zamislili smo si lasten protokol, po katerem lahko naprave pošiljajo vrednosti svojih meritev vozlišču, ki ima vlogo koorinatorja omrežja. Meritve smo pošiljali prek vgrajene konzole X-CTU nekajkrat na sekundo. Ker smo imeli na voljo samo dva modula in nobene zaprte naprave, ki komunicira po tej tehnologiji, bi bilo treba ta vtičnik dodatno testirati.

Vtičnik za protokol Z-Wave smo testirali na številnih napravah proizvajalcev Qubino. Naprave so se na ukaze odzivale pravilno in brez zapoznitev, ki se bile prisotne pri drugih predstavljenih knjižnicah. Z vtičnikom je bilo mogoče opravljati

osnovno funkcionalnost naprav, za večji nadzor nad njihovim delovanjem bi bilo treba vtičnik dopolniti z drugimi funkcionalnostmi (nastavljanje vrednosti konfiguracijskih parametrov). Pri testiranju vtičnika smo naleteli samo na eno napravo, ki je ni bilo mogoče pravilno vključiti v platformo zaradi nedokončanja začetnega intervjuja. Pri preostalih knjižnicah, ki so v celoti implementirane v programskem jeziku Java, smo opazili, da delujejo občutno počasneje, v določenih primerih pa se tudi zgodi, da naslovljene naprave ne prejmejo poslanih ukazov, kar smo preverili s posebnim ključem za spremljanje prometa v omrežjih Z-Wave. Delovanje vtičnika smo preizkusili z upravljalcema AEON LABS GEN5 in Sigma Controller (model ACC-UZB-E).

Pri naši arhitekturi smo za enoto z vlogo vmesnega vozlišča izbrali računalnik Raspberry Pi, ker zadnja različica že privzeto vsebuje podporo za brezžični internet in protokol Bluetooth. Namesto tega bi lahko izbrali druge, sorodne računalnike, kot na primer Beagle board black ali Odroid C2 (zadnji je po specifikacijah zmogljivejši od Raspberry Pi 3, nima pa vgrajene podpore za brezžični internet in protokol Bluetooth). Na tem računalniku tudi ni mogoče poganjati knjižnice Bluecove, ki je edina knjižnica, napisana v programskem jeziku Java, za protokol Bluetooth.

Poglavje 7

Zaključki in nadaljnje delo

Prednosti predstavljene arhitekture izhajajo iz uporabe platforme OM2M, ki se od drugih predstavljenih platform loči po tem, da ponuja standardizirani način komuniciranja med napravami po izbranem protokolu, omogoča pa tudi vključitev starejših naprav, za katere niso na voljo specifikacije standarda OneM2M. Platforma je iz izkušenj primernejša za tiste, ki so zelo domači v programskem jeziku Java in razvoju vtičnikov OSGi, in nikakor ni namenjena začetnikom v programiranju. Na uradni strani platforme sicer lahko najdemo nekaj vodičev, vendar je treba za njihovo popolno razumevanje poseči po specifikacijah standarda OneM2M. Drugi najpomembnejši vir informacij je uradni forum platforme, kjer njeni avtorji odgovarjajo na zastavljena vprašanja.

Platforma, ki smo jo razširili v okviru te naloge, ima številne možnosti uporabe. Glavni lastnosti platforme sta, da omogoča grajenje distribuiranega omrežja in da definira standardiziran način komunikacije po treh najpopularnejših protokolih. Napravo, na kateri bi tekla platforma OM2M, bi lahko recimo uporabili kot vmesno enoto za pošiljanje prejetih meritev naprav drugim (oblačnim) storitvam. Dva primera distribuiranih omrežij sta namestitev takšne vmesne enote v vsako sobo hotela ali v vsako sobo potniške ladje. V zadnjem primeru bi bila uporaba takšne enote veliko bolj smiselna v primerjavi z uporabo brezžičnih tehnologij, pri katerih se zaradi jeklene konstrukcije ladij pojavljajo motnje v komunikaciji. Arhitekturo bomo uporabili za razvoj enote za zajem medicinskih podatkov na daljavo, kar spada na področje telemedicine.

Literatura

[1] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, K. Drira, OM2M: Extensible ETSI-.compliant M2M Service Platform with Self-configuration Capability. *Procedia Computer Science*, pogl. 32, 2014, str. 1079–1086.

[2] LA. Grieco, M. Ben Alaya , T. Monteil, K. Drira, Architecting Information Centric ETSI–M2M systems. *IEEE International Conference on Pervasive Computing and Communications WIP*. Budapešta, Madžarska, marec 24–28 2014.

[3] M. Ben Alaya, S. Medijah, T. Monteil, K. Drira, Towards semantic interoperability in OneM2M architecture. *IEEE Communications magazine*, december 2015, str. 35–41.

[4] OneM2M TS-0001: Functional Architecture. Dostopno na: `http://www.onem2m.org/images/files/deliverables/TS-0001-Functional_Architecture-V1_13_1.pdf` (pridobljeno 22. september 2016)

[5] OneM2M TS-0004: Service Layer Core Protocol Specification. Dostopno na: `http://www.onem2m.org/images/files/deliverables/UpdateRelease1/TS-0004-Service_Layer_Core_Protocol-V1_6_0.zip` (pridobljeno 22. september 2016)

[6] OneM2M TS-0008: CoAP Protocol Binding. Dostopno na: `http://www.onem2m.org/images/files/deliverables/TS-0008-CoAP_Protocol_Binding-V1_3_2.pdf` (pridobljeno 22. september 2016)

[7] OneM2M TS-0009: HTTP Protocol Binding. Dostopno na: `http://www.onem2m.org/images/files/deliverables/TS-0009-HTTP_Protocol_Binding-V1_5_1.pdf` (pridobljeno 22. september 2016)

[8] OneM2M TS-0010: MQTT Protocol Binding. Dostopno na: `http://www.onem2m.org/images/files/deliverables/TS-0010-MQTT_Protocol_Binding-V1_5_1.pdf` (pridobljeno 22. september 2016)

[9] R. Sutaria, R. Govindachari, Making sense of interoperability: Protocols and Standardization initiatives in IOT, 2nd *International Workshop on Computing and Networking for Internet of Things*, 2013.

[10] P. Kess, H. Kropsu-Vehkapera, Standardization with IoT (internet-of-things) *Managing Innovation and Diversity in Knowledge Society Through Turbulent Time: Proceedings of the MakeLearn and TIIM Joint International Conference 2016* . Timisoara, Romunija, maj 25–27 2016.

[11] C. Pratt., F. Jansson, Encyclopedia of electronic componencts volume 3. *Maker Media*.

[12] RFC 7252 - The Constrained Application Protocol (CoAP) - IETF Tools. Dostopno na: `https://tools.ietf.org/html/rfc7252` (pridobljeno 22. september 2016)

[13] Robert Faludi, Building wireless sensor networks Založba O'Reilly Media. 2011.

[14] K. Paetz, Z-Wave Basics: Remote Control in Smart Homes. 2015

[15] Stran platforme OM2M. Dostopno na: `http://www.eclipse.org/om2m/` (obiskano 22. september 2016)

[16] Spletna storitev za prevajanje modulov mikrokontrolerjev NodeMCU. Dostopno na: `http://nodemcu-build.com/` (obiskano 22. september 2016)

[17] TMRh20. Dostopno na: `https://github.com/TMRh20/RF24` (obiskano 22. september 2016)

[18] Slika orodja Docker za delo z mikrokontrolerjem NodeMCU. Dostopno na: `https://hub.docker.com/r/marcelstoer/nodemcu-build/` (obiskano 22. september 2016)

[19] WZwave. Dostopno na: `https://github.com/whizzosoftware/WZWave` (obiskano 22. september 2016)

[20] Zwave library for java. Dostopno na: `https://github.com/oberasoftware/zwave` (obiskano 22. september 2016)

[21] Zwave4j. Dostopno na: `https://github.com/zgmnkv/zwave4j` (obiskano 22. september 2016)

[22] Paho. Dostopno na: `http://www.eclipse.org/paho/` (obiskano 22. september 2016)

[23] Posrednik Mosquitto. Dostopno na: `http://mosquitto.org/` (obiskano 22. september 2016)

[24] Posrednik Moquette. Dostopno na: `https://github.com/andsel/moquette` (obiskano 22. september 2016)

[25] Californium. Dostopno na: `https://github.com/eclipse/californium` (obiskano 22. september 2016)

[26] ESPlorer. Dostopno na: `http://esp8266.ru/esplorer/` (obiskano 22. september 2016)

[27] ESPFlasher. Dostopno na: `https://github.com/nodemcu/nodemcu-flasher` (obiskano 22. september 2016)

[28] Xbee-api. Dostopno na: `https://github.com/andrewrapp/xbee-api` (obiskano 22. september 2016)

[29] DHT. Dostopno na: `https://github.com/adafruit/DHT-sensor-library` (obiskano 22. september 2016)

[30] PubSubClient. Dostopno na: `https://github.com/knolleary/pubsubclient` (obiskano 22. september 2016)

[31] OpenHAB. Dostopno na: `http://www.openhab.org/` (obiskano 22. september 2016)

[32] HomeGenie. Dostopno na: `http://www.homegenie.it/` (obiskano 22. september 2016)

[33] HomeAssistant. Dostopno na: `https://home-assistant.io/` (obiskano 22. september 2016)

[34] Domoticz. Dostopno na: https://domoticz.com/ (obiskano 22. september 2016)

[35] OpenZWave. Dostopno na: http://www.openzwave.com/ (obiskano 22. september 2016)

[36] MySensors. Dostopno na: https://www.mysensors.org/ (obiskano 22. september 2016)

[37] Serial nRF24L01. Dostopno na: https://github.com/nkolban/RF24Serial/ (obiskano 22. september 2016)

[38] Dokumentacija platforme OpenZWave. Dostopno na: http://www.openzwave.com/dev/ (obiskano 22. september 2016)

[39] Stran standarda oBIX. Dostopno na: http://www.obix.org/ (obiskano 22. september 2016)

[40] Sigma USB controller. Dostopno na: http://z-wave.sigmadesigns.com/products#UZB (obiskano 22. september 2016)

[41] Aeon GEN5 ali Series 2 USB controller. Dostopno na: http://aeotec.com/z-wave-usb-stick (obiskano 22. september 2016)

[42] RaZberry. Dostopno na: http://razberry.z-wave.me/index.php?id=9 (obiskano 22. september 2016)

[43] UZB. Dostopno na: https://www.z-wave.me/index.php?id=28 (obiskano 22. september 2016)

[44] Pravilna inicializacija knjižnice Zwave4j v primeru vtičnika OSGi. Dostopno na: https://github.com/zgmnkv/zwave4j/issues/36 (obiskano 22. september 2016)

[45] Prenos, prevajanje in konfiguracija platforme OM2M na napravah Windows. Dostopno na: https://wiki.eclipse.org/OM2M/one#Introduction (obiskano 22. september 2016)

[46] Navodila za izdelavo lastnega vtičnika IPU. Dostopno na: https://wiki.eclipse.org/OM2M/one/Developer (obiskano 22. september 2016)

[47] Spletna stran podjetja Digi. Dostopno na: `http://www.digi.com/` (obiskano 22. september 2016)

[48] Pepper1: `http://www.pepper1.net/zwavedb/` (obiskano 22. september 2016)

[49] Knjižnica ZwaveMe: `https://z-wave.me/` (obiskano 22. september 2016)

[50] Primer uporabe knjižnice Californium na pametnih telefonih: `https://github.com/Tanganelli/californium/tree/master/cf-android` (obiskano 22. september 2016)

[51] Nastavljanje knjižnice RXTX na napravah Windows: `https://www.youtube.com/watch?v=43Vdpz1YmdU` (obiskano 22. september 2016)

[52] Odjemalec za protokol CoAP - Copper: `https://addons.mozilla.org/en-US/firefox/addon/copper-270430/` (obiskano 22. september 2016)

[53] Odjemalec za protokol CoAP - Libcoap: `https://libcoap.net/` (obiskano 22. september 2016)

[54] Odjemalec za protokol HTTP - Advanced rest client: `https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo` (obiskano 22. september 2016)

[55] Knjižnica za delo s protokolom HTTP - Apache HTTPClient: `https://hc.apache.org/httpcomponents-client-ga/` (obiskano 22. september 2016)

[56] Pregled zvez in konzorcijev na področju interneta stvari: `http://postscapes.com/internet-of-things-alliances-roundup/` (obiskano 22. september 2016)

[57] Stran z namestitvenimi datotekami posrednika Mosquitto: `https://mosquitto.org/download/` (obiskano 22. september 2016)

[58] Stran knjižnice OpenSSL: `http://slproweb.com/products/Win32OpenSSL.html` (obiskano 22. september 2016)

[59] Stran knjižnice Pthreads: `http://slproweb.com/products/Win32OpenSSL.html` (obiskano 22. september 2016)

Poglavje 8

Priloge

A Seznam razredov tehnologije Z-Wave

V spodnji tabeli je predstavljen nabor razredov tehnologije Z-Wave [14], ki jih naprave lahko podpirajo. Razredi imajo lahko več različic in morajo biti med seboj združljivi za nazaj. Vsaka naprava mora obvezno implementirati razred COMMAND_CLASS_BASIC, ker služi za upravljanje osnovne funkcionalnosti naprave.

Oznaka	**Ime razreda**	**Opis**
0x71	ALARM	Razred implementirajo predvsem naprave, ki zaznavajo specifične prožene dogodke, ki zadevajo varnost (prisotnost dima, požar, poplava, gibanje). S tem razredom lahko naprave obvestijo druge naprave o proženih dogodkih. Običajno eksplicitno ne kličemo metod tega razreda, ampak prejemamo asinhrona poročila ob proženju določenega dogodka.
0x9C	ALARM_SENSOR	Konfiguriranje naprav, ki podpirajo razred ALARM.
0x9D	ALARM_SILENCE	Se uporablja za začasno izključevanje alarma, če se je ta sprožil.

0x22	APPLICATION STATUS	Splošne metode, ki se uporabljajo za zagotavljanje optimalnega delovanja omrežja Z-Wave.
0x85	ASSOCIATION	Asociacije so mehanizem, ki napravam omogoča upravljanje drugih naprav. S tem razredom lahko nastavljamo, brišemo in pridobivamo njihova stanja na izbrani napravi. Z nastavljanjem asociacij lahko na primer nastavimo, da se ob prihodu domov (prijava našega telefona v domače brezžično omrežje) prižge luč v avli hiše (asociacija na luč v avli).
0x9B	ASSOCIATION COMMAND CONFIGURATION	Konfiguriranje naprav, ki podpirajo razred ASSOCIATION.
0x20	BASIC	Krmiljenje osnovne funkcionalnosti naprave. Običajno ta razred samo preslika funkcionalnost nekega drugega razreda, ki ga naprava implementira. Primer osnovne funkcionalnosti naprave bi bilo upravljanje žaluzij ali prižiganje luči.
0x80	BATTERY	Pridobimo informacijo o stanju baterije na napravi.
0x46	CLIMATE CONTROL SCHEDULE	Proženje gretja ali hlajenja klimatske naprave z dvema vnaprej definiranima vrednostima.
0x81	CLOCK	Razred omogoča implementacijo ure na napravi.

0x70	CONFIGURATION	Nastavljanje vrednosti konfiguracijskih parametrov. Konfiguracijski parametri omogočajo spreminjanje privzetega delovanja naprave. Definira jih proizvajalec naprav. Vsaka naprava, ki jo knjižnica OpenZwave podpira, mora imeti lastno datoteko XML, v kateri so predstavljeni konfiguracijski parametri in vrednosti, ki jih zadnji lahko zavzamejo.
0x62	DOORLOCK	Delo z vratnimi ključavnicami. Poleg metode za zaklepanje in odklepanje vsebuje tudi metodo za nastavljanje zapoznitve določene operacije.
0x4C	DOOR LOCK LOGGING	Beleženje zgodovine proženih dogodkov, ki so se zgodili na napravi.
0x7A	FIRMWARE UPDATE MD	Pridobivanje podatkov o trenutno nameščeni programski opremi. Posodabljanje programske opreme na daljavo (angl. *Over the air - OTA*)
0x82	HAIL	Omogoča napravam, da poročajo upravljalcem o dogodkih, ki so se zgodili.
0x87	INDICATOR	Prikazovanje trenutnega stanja naprave.
0x72	MANUFACTURER SPECIFIC	Razred, s katerim lahko pridobimo podatke o proizvajalcu naprave. Podatki, ki jih s tem razredom lahko pridobimo, so: enolični identifikator proizvajalca, enolični identifikator naprave, serijska številka naprave (če to modul podpira), različica nameščene programske opreme.

0x32	METER	Če nas zanima poraba naših naprav, jo lahko pridobimo s tem razredom. Z nastavljanjem parametra 'scale' lahko zahtevamo vrednosti za različne merilne enote (moč, voltaža, tok).
0x35	METER PULSE	Implementacija števcev za merjenje porabe različnih količin (voda, elektrika, plin).
0x3D	METER TABLE MONITOR	Definira metode za branje preteklih in akumuliranih vrednosti števcev za različne merljive količine.
0x60	MULTI CHANNEL	Naprave Z-Wave podpirajo delovanje na več kanalih (angl. *endpoints*). Ta razred omogoča pošiljanje ukazov na izbrani kanal. Število kanalov je odvisno od implementacije.
0x8E	MULTI CHANNEL ASSOCIATION	Enaka funkcionalnost kot pri razredu ASSOCIATION, narejena za upravljanje drugih (ne osnovnega) kanalov naprave.
0x8F	MULTI CMD	Enkapsulacija večjega števila ukazov v en paket.
0x77	NODE NAMING	Uporabniku omogoča, da napravi nastavi novo ime, ki si ga bo lažje zapomnil.
0x00	NO OPERATION	Preverjanje dosegljivosti izbranega vozlišča.
0x73	POWERLEVEL	Nastavljanje moči oddajanja naprave, kar pride prav pri njenem testiranju in nameščanju.
0x75	PROTECTION	Varnostni razred, ki preprečuje nenamerno uporabo.

0x2B	SCENE ACTIVATION	Scenariji predstavljajo zbirko dogodkov, ki se prožijo ob definiranih pogojih na izbranih napravah. Ta razred omogoča proženje scenarijev. Scenariji omogočajo istočasno upravljanje večjega števila naprav. Z enim scenarijem bi lahko nastavili, da bi se na primer ob šestih zjutraj dviginile vse žaluzije.
0x2C	SCENE ACTUATOR CONF	Omogoča nastavljanje dogodkov in pogojev scenarijev na določeni napravi.
0x2D	SCENE CONTROLLER CONF	Povezuje kanal in izbrani scenarij.
0x4E	SCHEDULE ENTRY LOCK	Omogoča zaklepanje ključavnice ob vnaprej določenem času.
0x98	SECURITY	Naprave Z-Wave podpirajo dve različici razreda Security (S1 in S2). Glavna razlika med njima je, da je treba pri napravah, ki implementirajo različico S2, na vsaki napravi vpisovati dodatno kodo PIN ob vključevanju naprave v omrežje. Naprave, ki implementirajo ta razred, morajo delovati tudi na upravljalcih, ki tega razreda ne implementirajo.
0x30	SENSOR BINARY	Razred omogoča spremljanje stanj vhodnih naprav (na primer stanje stikala, ki je priklopljeno na napravo).
0x9E	SENSOR CONFIGURATION	Omogoča nastavljanje pragov, ob katerih bodo senzorji poročali svoje meritve.
0x31	SENSOR MULTILEVEL	Branje vrednosti s senzorja, ki lahko vrača vrednosti iz določenega intervala (na primer za branje vrednosti temperaturnega senzorja).

0x94	SIMPLE AV CONTROL	Pošiljanje enostavnih ukazov vozliščem, ki predstavljajo avdio- in videonaprave.
0x27	SWITCH ALL	Upravljanje večjega števila naprav. Razred vsebuje metode, s katerimi lahko določimo, katere naprave se bodo odzivale na ukaze tega razreda.
0x25	SWITCH BINARY	Če lahko napravo upravljamo z dvema stanjema, lahko uporabimo ta razred.
0x26	SWITCH MULTILEVEL	Če lahko napravo upravljamo z intervalom vrednosti (na primer zatemnilniki), lahko uporabimo ta razred.
0x44	THERMOSTAT FAN MODE	Nastavljanje načina delovanja ventilacije.
0x40	THERMOSTAT MODE	Nastavljanje različnih načinov delovanja termostata (gretje, hlajenje).
0x43	THERMOSTAT SETPOINT	Omogoča nastavljanje izbrane vrednosti temperature (angl. *setpoint*), ki jo bo termostat poskušal vzdrževati.
0x8A	TIME	Pridobivanje datuma in časa, ki sta zabeležena na napravi.
0x8B	TIME PARAMETERS	Nastavljanje datuma in časa na napravi.
0x63	USER CODE	Razred za upravljanje uporabniških gesel.
0x86	VERSION	Vsak razred, ki ga naprava podpira, ima lahko več različic, ki morajo biti med seboj združljive za nazaj. S tem razredom lahko pridobimo različice razredov, ki jih naprava implementira.
0x84	WAKE_UP	Omogoča napravam, ki se napajajo prek baterij, da svoje upravljalce obvestijo o tem, da so zbujene ter da lahko sprejemajo sporočila.

B Opombe pri razvoju vtičnikov IPU

V tem poglavju želimo omeniti nekaj stvari, ki so v vodičih izpuščene, vendar jih je pri razvoju vtičnikov (in pri delu z platformo OM2M) dobro vedeti:

- Pri ustvarjanju novega vtičnika je priporočeno, da novi vtičnik shranimo v glavno mapo platforme (org.eclipse.om2m), kjer so že preostali vtičniki. To je potrebno zaradi enostavnejše migracije celotnega projekta, pa tudi zaradi sklicevanja na vire z uporabo relativnih poti.

- Po ustvarjanju vtičnika se nam lahko na napačnem mestu ustvari začetni razred Activator, ki je namenjen zagonu in zaustavitvi vtičnika. To lahko preprečimo tako, da v čarovniku za izdelavo novega vtičnika odkljukamo možnost "Generate an activator, a Java class, that controls the plug-in's life cycle".

- Priporočeno razvojno okolje za razvoj vtičnikov je orodje Eclipse. To orodje sprememb v konfiguracijskih datotekah ne shranjuje samodejno (spremembe se označujejo z znakom *), zato je treba biti pozoren, da vsakokrat ročno shranimo spremembe s kombinacijo tipk Ctrl+S.

- Med izdelavo vtičnika je treba navesti različne module, ki jih vtičnik potrebuje za svoje delovanje. Če določenega modula med iskanjem ne najdemo, najprej preverimo, ali smo v orodje uvozili vse gnezdene projekte platforme.

- Vsakemu vtičniku je mogoče nastaviti samodejen način zagona. To storimo tako, da v datoteki om2m.product pripadajoče enote (IN-CSE, MN-CSE, ASN-CSE), na kateri vtičnik teče, spremenimo lastnost Auto-start. V naši različici dopolnjene platforme OM2M sta dva vtičnika za protokol CoAP. Vtičnik, ki je privzeto že vključen v platformo, se samodejno zažene ob zagonu izbrane enote. Ker oba vtičnika uporabljata enako knjižnico, ne moremo poganjati obeh naenkrat. Če zaženemo drugi vtičnik, bo javljena napaka.

- Pri knjižnicah, ki uporabljajo serijsko komunikacijo prek izbranega serijskega vmesnika (vrat COM), se običajno na napravah Windows uporablja knjižnica RXTX. Za uporabo te knjižnice je potrebnih nekaj dodatnih nastavitev [51].

- Starejše različice platforme OM2M, ki implementirajo standard OneM2M, definirajo polje v glavi zahtevka (pri protokolih HTTP in CoAP), s katerim lahko določijo ime novega vira, ki bo ustvarjen. To ni povsem po specifikacijah standarda OneM2M, zato se priporoča uporaba atributa "rn" oziroma "nm" tako, kot je predstavljeno v poglavju 4.5.

- Pred prevajanjem platforme in njenih vtičnikov je treba zapreti morebitne odprte instance platforme, sicer postopek prevajanja ne bo uspešen.

- V spletnem vmesniku platforme so imena atributov vsebnikov in njihovih instanc predstavljena s kratkimi imeni. Razlago določenega atributa najdemo v tabelah od 8.2.3-1 do 8.2.3-5 [5].

C Seznam odprtokodnih platform

Naziv platforme	**Povezava**
Ag Control	http://www.agocontrol.com/
Domogik	http://www.domogik.org/en/en
Domoticz	http://www.domoticz.com/
DomotiGa	http://www.domotiga.nl/
FHEM	http://fhem.de/
Freedomotic	http://www.freedomotic.com/
Homegenie	http://www.homegenie.it/
Homidom	http://www.homidom.com/
Housecream	http://housecream.org/
MisterHouse	http://misterhouse.sourceforge.net/
Nodo	http://www.nodo-domotica.nl/
OpenDomo	http://en.opendomo.org/
OpenHAB	https://code.google.com/p/openhab/
OpenRemote	http://www.openremote.org/
OpenSourceAutomation	http://www.opensourceautomation.com
PiDome	http://pidome.wordpress.com/
PiHome	http://pihome.harkemedia.de/
Pilight	http://www.pilight.org/
Pytomation	http://www.pytomation.com/
QHom Automation	http://qwhomeautomation.com/
Rasp485berry	http://rasp485berry.wordpress.com/
Wiseflat	http://wiseflat.com/
ZVirtualScenes	https://code.google.com/p/zvirtualscenes/

Tabela 8.2: Seznam odprtokodnih platform.